US009311141B2

(12) **United States Patent**
Wheeler et al.

(10) **Patent No.: US 9,311,141 B2**
(45) **Date of Patent: Apr. 12, 2016**

(54) **SURVIVAL RULE USAGE BY SOFTWARE AGENTS**

(75) Inventors: **Thomas T. Wheeler**, Frisco, TX (US); **John P. Patoskie**, Allen, TX (US)

(73) Assignee: **CALLAHAN CELLULAR L.L.C.**, Wilmington, DE (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1053 days.

(21) Appl. No.: **13/429,785**

(22) Filed: **Mar. 26, 2012**

(65) **Prior Publication Data**

US 2012/0185826 A1 Jul. 19, 2012

Related U.S. Application Data

(63) Continuation of application No. 11/645,198, filed on Dec. 22, 2006, now abandoned.

(51) **Int. Cl.**

G06N 5/04 (2006.01)
G06F 9/48 (2006.01)
G06F 11/30 (2006.01)

(52) **U.S. Cl.**

CPC ***G06F 9/4862*** (2013.01); ***G06N 5/04*** (2013.01); *G06F 11/3089* (2013.01)

(58) **Field of Classification Search**

CPC G06N 5/04
USPC 706/45
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,714,995	A	12/1987	Materna et al.
5,465,111	A	11/1995	Fukushima et al.
5,495,533	A	2/1996	Linehan et al.
5,524,253	A	6/1996	Pham et al.
5,604,800	A	2/1997	Johnson et al.
5,675,711	A	10/1997	Kephart et al.
5,721,912	A	2/1998	Stepczyk et al.
5,778,395	A	7/1998	Whiting et al.
5,812,997	A	9/1998	Morimoto et al.
5,859,911	A	1/1999	Angelo et al.
5,930,798	A	7/1999	Lawler et al.
5,982,891	A	11/1999	Ginter et al.
6,038,500	A	3/2000	Weiss
6,055,562	A	4/2000	Devarakonda et al.
6,065,040	A	5/2000	Mima et al.
6,088,689	A	7/2000	Kohn et al.
6,113,650	A	9/2000	Sakai
6,134,580	A	10/2000	Tahara et al.
6,134,660	A	10/2000	Boneh et al.
6,154,777	A	11/2000	Ebrahim
6,172,986	B1	1/2001	Watanuki et al.

(Continued)

FOREIGN PATENT DOCUMENTS

JP	2005190103	7/2005

OTHER PUBLICATIONS

Bursell et al ("Comparison of autonomous mobile agent technologies" 1997).*

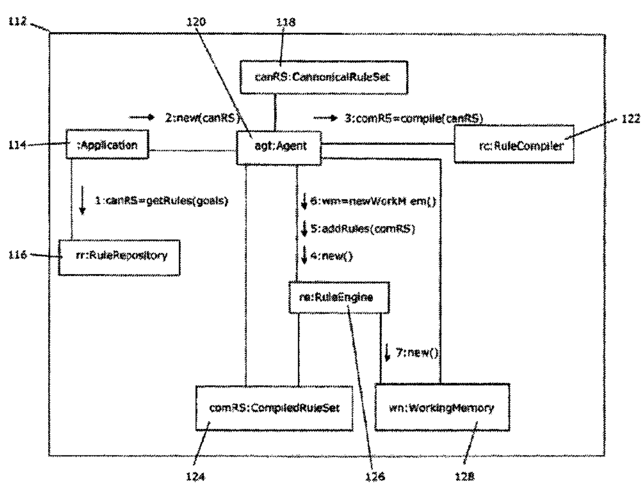
(Continued)

Primary Examiner — Lut Wong

(57) **ABSTRACT**

A method for determining the lifespan of an agent utilizing a rule engine and a set of canonical survival rules, in an execution environment comprising collecting a survival rule, asserting a survival data into a working memory and executing the rule engine with the set of survival rules and the working memory.

17 Claims, 25 Drawing Sheets



(56) **References Cited**

U.S. PATENT DOCUMENTS

6,199,195	B1	3/2001	Goodwin et al.
6,202,060	B1	3/2001	Tran
6,209,124	B1	3/2001	Vermeire et al.
6,230,309	B1	5/2001	Turner et al.
6,233,601	B1	5/2001	Walsh
6,256,773	B1	7/2001	Bowman-Amuah
6,272,478	B1	8/2001	Obata et al.
6,282,563	B1	8/2001	Yamamoto et al.
6,282,697	B1	8/2001	Fables et al.
6,304,556	B1	10/2001	Haas
6,330,677	B1	12/2001	Madoukh
6,343,265	B1	1/2002	Glebov et al.
6,349,343	B1	2/2002	Foody et al.
6,389,462	B1	5/2002	Cohen et al.
6,438,744	B2	8/2002	Toutonghi et al.
6,442,537	B1	8/2002	Karch
6,452,910	B1	9/2002	Vij et al.
6,477,372	B1	11/2002	Otting
6,496,871	B1	12/2002	Jagannathan et al.
6,600,430	B2	7/2003	Minagawa et al.
6,600,975	B2	7/2003	Moriguchi et al.
6,611,516	B1	8/2003	Pirkola et al.
6,629,032	B2	9/2003	Akiyama
6,629,128	B1	9/2003	Glass
6,662,642	B2	12/2003	Breed et al.
6,671,680	B1	12/2003	Iwamoto et al.
6,687,761	B1	2/2004	Collins et al.
6,701,381	B2	3/2004	Hearne et al.
6,714,844	B1	3/2004	Dauner et al.
6,738,975	B1	5/2004	Yee et al.
6,744,352	B2	6/2004	Lesesky et al.
6,744,358	B1	6/2004	Bollinger
6,744,820	B1	6/2004	Khairallah et al.
6,768,417	B2	7/2004	Kuragaki et al.
6,816,452	B1	11/2004	Maehata
6,831,375	B1	12/2004	Currie et al.
6,851,108	B1	2/2005	Syme
6,862,443	B2	3/2005	Witte
6,876,845	B1	4/2005	Tabata et al.
6,879,570	B1	4/2005	Choi
6,895,391	B1	5/2005	Kausik
6,901,588	B1	5/2005	Krapf et al.
6,904,593	B1	6/2005	Fong et al.
6,931,455	B1	8/2005	Glass
6,931,623	B2	8/2005	Vermeire et al.
6,947,965	B2	9/2005	Glass
6,951,021	B1	9/2005	Bodwell et al.
6,957,439	B1	10/2005	Lewallen
6,963,582	B1	11/2005	Xu
6,981,150	B2	12/2005	Little et al.
6,985,929	B1	1/2006	Wilson et al.
6,993,774	B1	1/2006	Glass
7,010,689	B1	3/2006	Matyas, Jr. et al.
7,043,522	B2	5/2006	Olson et al.
7,047,518	B2	5/2006	Little et al.
7,055,153	B2	5/2006	Beck et al.
7,058,645	B2	6/2006	Seto et al.
7,062,708	B2	6/2006	Mani et al.
7,069,551	B2	6/2006	Fong et al.
7,127,259	B2	10/2006	Ueda et al.
7,127,724	B2	10/2006	Lewallen
7,146,614	B1	12/2006	Nikols et al.
7,146,618	B1	12/2006	Mein et al.
7,172,113	B2	2/2007	Olenick et al.
7,174,533	B2	2/2007	Boucher
7,197,742	B2	3/2007	Arita et al.
7,210,132	B2	4/2007	Rivard et al.
7,213,227	B2	5/2007	Kompalli et al.
7,225,425	B2	5/2007	Kompalli et al.
7,228,141	B2	6/2007	Sethi
7,231,403	B1	6/2007	Howitt et al.
7,237,225	B2	6/2007	Kompalli et al.
7,293,261	B1	11/2007	Anderson et al.
7,376,959	B2	5/2008	Warshavsky et al.
7,475,107	B2	1/2009	Maconi et al.
7,477,897	B2	1/2009	Bye
7,496,637	B2	2/2009	Han et al.
7,499,990	B1	3/2009	Tai et al.
7,502,615	B2	3/2009	Wilhoite et al.
7,506,309	B2	3/2009	Schaefer
7,519,455	B2	4/2009	Weiss
7,566,002	B2	7/2009	Love et al.
7,643,447	B2	1/2010	Watanuki et al.
7,660,777	B1	2/2010	Hauser
7,660,780	B1	2/2010	Patoskie
7,664,721	B1	2/2010	Hauser
7,698,243	B1	4/2010	Hauser
7,702,602	B1	4/2010	Hauser
7,702,603	B1	4/2010	Hauser
7,702,604	B1	4/2010	Hauser
7,774,789	B1	8/2010	Wheeler
7,810,140	B1	10/2010	Lipari et al.
7,823,169	B1	10/2010	Wheeler
7,840,513	B2	11/2010	Hauser
7,844,759	B1	11/2010	Cowin
2001/0029526	A1	10/2001	Yokoyama et al.
2001/0051515	A1	12/2001	Rygaard
2001/0056425	A1	12/2001	Richard
2002/0016912	A1	2/2002	Johnson
2002/0032783	A1	3/2002	Tuatini
2002/0035429	A1	3/2002	Banas
2002/0091680	A1	7/2002	Hatzis et al.
2002/0115445	A1	8/2002	Myllymaki
2002/0116454	A1	8/2002	Dyla et al.
2002/0159479	A1	10/2002	Watanuki et al.
2002/0174222	A1	11/2002	Cox
2002/0196771	A1	12/2002	Vij et al.
2003/0009539	A1	1/2003	Hattori
2003/0018950	A1	1/2003	Sparks et al.
2003/0023573	A1	1/2003	Chan et al.
2003/0033437	A1	2/2003	Fischer et al.
2003/0046377	A1	3/2003	Daum et al.
2003/0046432	A1	3/2003	Coleman et al.
2003/0051172	A1	3/2003	Lordermann et al.
2003/0055898	A1	3/2003	Yeager et al.
2003/0070071	A1	4/2003	Riedel et al.
2003/0101441	A1	5/2003	Harrison et al.
2003/0177170	A1	9/2003	Glass
2003/0191797	A1	10/2003	Gurevich et al.
2003/0191969	A1	10/2003	Katsikas
2003/0225789	A1	12/2003	Bussler et al.
2003/0225935	A1	12/2003	Rivard et al.
2004/0003243	A1	1/2004	Fehr et al.
2004/0015539	A1	1/2004	Alegria et al.
2004/0037315	A1	2/2004	Delautre et al.
2004/0044985	A1	3/2004	Kompalli et al.
2004/0044986	A1	3/2004	Kompalli et al.
2004/0044987	A1	3/2004	Kompalli et al.
2004/0064503	A1	4/2004	Karakashian et al.
2004/0078423	A1	4/2004	Satyavolu et al.
2004/0078687	A1	4/2004	Partamian et al.
2004/0082350	A1	4/2004	Chen et al.
2004/0088369	A1	5/2004	Yeager et al.
2004/0111730	A1	6/2004	Apte
2004/0133656	A1	7/2004	Butterworth et al.
2004/0148073	A1	7/2004	Hawig et al.
2004/0153653	A1	8/2004	Abhyankar et al.
2004/0172614	A1	9/2004	Gallagher
2004/0194072	A1	9/2004	Venter
2004/0203731	A1	10/2004	Chen et al.
2004/0220952	A1	11/2004	Cheenath
2004/0221292	A1	11/2004	Chiang et al.
2005/0004727	A1	1/2005	Remboski et al.
2005/0030202	A1	2/2005	Tsuboi
2005/0090279	A9	4/2005	Witkowski et al.
2005/0114832	A1	5/2005	Manu
2005/0141706	A1	6/2005	Regli et al.
2005/0144218	A1	6/2005	Heintz
2005/0163549	A1	7/2005	Shima et al.
2005/0172123	A1	8/2005	Carpenter et al.
2005/0246302	A1	11/2005	Lorenz et al.
2005/0256614	A1	11/2005	Habermas
2005/0256876	A1	11/2005	Eidson
2005/0262155	A1	11/2005	Kress et al.

(56) **References Cited**

U.S. PATENT DOCUMENTS

2005/0273668	A1	12/2005	Manning
2005/0281363	A1	12/2005	Qi et al.
2006/0005177	A1	1/2006	Atkin et al.
2006/0031850	A1	2/2006	Falter et al.
2006/0040640	A1	2/2006	Thompson et al.
2006/0041337	A1	2/2006	Augsburger et al.
2006/0048145	A1	3/2006	Celli et al.
2006/0080646	A1	4/2006	Aman
2006/0090103	A1	4/2006	Armstrong et al.
2006/0111089	A1	5/2006	Winter et al.
2006/0112183	A1	5/2006	Corson et al.
2006/0121916	A1	6/2006	Aborn
2006/0123396	A1	6/2006	Fox et al.
2006/0143600	A1	6/2006	Cottrell et al.
2006/0149746	A1	7/2006	Bansod et al.
2006/0167981	A1	7/2006	Bansod et al.
2006/0173857	A1	8/2006	Jackson
2006/0190931	A1	8/2006	Scott et al.
2006/0200494	A1	9/2006	Sparks
2006/0206864	A1	9/2006	Shenfield et al.
2006/0220900	A1	10/2006	Ceskutti et al.
2006/0221901	A1	10/2006	Yaqub et al.
2006/0245406	A1	11/2006	Shim
2006/0256008	A1	11/2006	Rosenberg
2006/0272002	A1	11/2006	Wightman et al.
2007/0004432	A1	1/2007	Hwang et al.
2007/0015495	A1	1/2007	Winter et al.
2007/0026871	A1	2/2007	Wager
2007/0103292	A1	5/2007	Burkley et al.
2007/0112773	A1	5/2007	Joyce
2007/0223432	A1	9/2007	Badarinath
2007/0243880	A1	10/2007	Gits et al.
2008/0077667	A1	3/2008	Hwang et al.
2010/0161543	A1	6/2010	Hauser
2010/0223210	A1	9/2010	Patoskie
2010/0235459	A1	9/2010	Wheeler

OTHER PUBLICATIONS

Non Final Office Action, U.S. Appl. No. 11/052,680 (Nov. 19, 2010), 17 pages.

Non Final Office Action, U.S. Appl. No. 11/645,197, (Nov. 29, 2010), 20 pages.

Notice of Allowance, U.S. Appl. No. 12/647,964 (Nov. 30, 2010),7 pages.

Non-Final Office Action, U.S. Appl. No. 12/647,964, (Sep. 8, 2010), 5 pages.

Final Office Action, U.S. Appl. No. 10/975,827 (Sep. 15, 2010), 22 pages.

Final Office Action, U.S. Appl. No. 11/645,192, (Sep. 15, 2010),14 pages.

Erfurth, Christian et al., Migration Intelligence for Mobile Agents, (2001), 8 pages.

Final Office Action, U.S. Appl. No. 11/645,194 (Sep. 15, 2010),28 pages.

Final Office Action, U.S. Appl. No. 11/645,196, (Sep. 22, 2010), 33 pages.

Notice of Allowance, U.S. Appl. No. 12/714,753 (Sep. 27, 2010),7 pages.

Advisory Action, U.S. Appl. No. 11/645,197, (Oct. 6, 2010), 3 pages.

Notice of Allowance, U.S. Appl. No. 11/086,121 (Oct. 7, 2010),6 pages.

Notice of Allowance, U.S. Appl. No. 11/645,195, (Oct. 7, 2010), 8 pages.

Final Office Action, Application Oct. 21, 2010 ,34 pages.

Final Office Action, U.S. Appl. No. 11/645,191, (Oct. 28, 2010), 19 pages.

Final Office Action, U.S. Appl. No. 11/086,102 (Jul. 1, 2010), 14 pages.

Notice of Allowability, U.S. Appl. No. 10/975,146 (Jul. 13, 2010),2 pages.

Rosenberg, Jonathan et al., "Programming Internet Telephony Services", IEEE Network, (May/Jun. 1999), 8 pages.

Notice of Allowance, U.S. Appl. No. 11/439,059, (Jul. 14, 2010), 12 pages.

Final Office Action, U.S. Appl. No. 11/645,197 (Jul. 19, 2010), 16 pages.

Advisory Action, U.S. Appl. No. 11/052,680, (Jul. 28, 2010), 3 pages.

Advisory Action, U.S. Appl. No. 11/645,303, (Jul. 28, 2010), 3 pages.

"The Sims", Sims, Electranic Arts, (2000), 57 pages.

Taylor, TL "Does WoW Change Everything?", Games and Culture, vol. 1, No. 4, (Oct. 2006), 20 pages.

Barba, Rick "The Sims Prima's Official Strategy Guide", Prima Publishing: ISBN: 7615-2339-1 (2000),98 Pages.

Hinchey, Michael G., et al., "Formal Approaches to Agent-Based Systems", 2nd International Workshop FAABS, (Oct. 2002),291 pages.

Non Final Office Action, U.S. Appl. No. 11/645,301 (Jul. 30, 2010), 18 pages.

Non Final Office Action, Application 11/1645,303, (Aug. 19, 2010), 20 pages.

Chang, Yao-Chung et al., "All-IP Convergent Communications over Open Service Architecture", 2005 Wireless Telecommunications Symposium, IEEE, 0-7803-8856, (2005), pp. 202-210.

"WOW, World of Warcraft", Wikipedia (online), Retrieved from Internet in 2010, <URL:htto:llen.wikioedia.orq/wiki/World of Warcraft> (2010), 28 pages.

Notice of Allowance, U.S. Appl. No. 11/495,053 (Aug. 31, 2010),7 pages.

Advisory Action, U.S. Appl. No. 11/086,120, (May 4, 2010), 3 pages.

Advisory Action, U.S. Appl. No. 11/495,053 (May 26, 2010),3 pages.

Final Office Action, U.S. Appl. No. 11/086,121, (Jun. 16, 2010), pp. 1-30.

Final Office Action, U.S. Appl. No. 11/086,120 (Feb. 23, 2010),43 pages.

Final Office Action, U.S. Appl. No. 11/495,053, (Mar. 17, 2010), 17 pages.

Final Office Action, U.S. Appl. No. 11/052,680 (May 17, 2010), 12 pages.

Final Office Action, U.S. Appl. No. 11/645,301 May 12, 2010), 32 pages.

Final Office Action, U.S. Appl. No. 11/645,303, May 21, 2010), 32 pages.

Non Final Office Action, U.S. Appl. No. 10/975,827 Mar. 25, 2010, 19 pages.

Non Final Office Action, U.S. Appl. No. 11/439,059, Mar. 26, 2010,6 pages.

Non Final Office Action, U.S. Appl. No. 11/645,191 (May 5, 2010), 17 pages.

Non Final Office Action, U.S. Appl. No. 11/645,192, May 4, 2010, 14 pages.

Non Final Office Action, U.S. Appl. No. 11/645,194, Mar. 26, 2010, 26 pages.

Non Final Office Action, U.S. Appl. No. 11/645,196, Apr. 28, 2010, 29 pages.

Non Final Office Action, U.S. Appl. No. 11/645,200 May 26, 2010, 34 pages.

Non-Final Office Action, U.S. Appl. No. 11/052,680, Feb. 4, 2010, 12 pages.

Non-Final Office Action, U.S. Appl. No. 11/086,101 Sep. 2, 2009 ,20 pages.

Non-Final Office Action, U.S. Appl. No. 11/086,102, Mar. 5, 2010 , 15 pages.

Non-Final Office Action, U.S. Appl. No. 11/086,121 Dec. 30, 2009, 25 pages.

Non-Final Office Action, U.S. Appl. No. 11/439,059, Oct. 30, 2009, 9 pages.

Non-Final Office Action, U.S. Appl. No. 11/495,053 Jun. 10, 2009, 13 pages.

Non-Final Office Action, U.S. Appl. No. 11/645,195, Apr. 15, 2010, 20 pages.

Non-Final Office Action, U.S. Appl. No. 11/645,197 Feb. 24, 2010, 13 pages.

(56) **References Cited**

OTHER PUBLICATIONS

Non-Final Office Action, U.S. Appl. No. 11/645,301, Jan. 6, 2010, 25 pages.
Non-Final Office Action, U.S. Appl. No. 11/645,303 Feb. 8, 2010, 18 pages.
Notice of Allowance, U.S. Appl. No. 10/975,623, Feb. 23, 2010), 9 pages.
Notice of Allowance, U.S. Appl. No. 11/086,101 Feb. 25, 2010), 9 pages.
Notice of Allowance, U.S. Appl. No. 11/645,292, Nov. 17, 2009, 9 pages.
Notice of Allowance, U.S. Appl. No. 11/645,300 Nov. 18, 2009, 9 pages.
Notice of Allowance, U.S. Appl. No. 11/645,302, Nov. 17, 2009, 8 pages.
Notice of Allowance, U.S. Appl. No. 11/645,306 Nov. 17, 2009, 8 pages.
Restriction Requirement, U.S. Appl. No. 11/645,191, Jan. 28, 2010, 7 pages.
Restriction Requirement, U.S. Appl. No. 11/645,192 Jan. 28, 2010 ,6 pages.
Restriction Requirement, U.S. Appl. No. 11/645,200, Feb. 26, 2010 ,8 pages.
Bagci, et al., "Communication and Security Extensions for a Ubiquitous Mobile Agent System (UbiMAS)", In Proceedings of CF 2005, Available at <http://portal.acm.orglft_gateway.cfm?id=1062302&type=pdf&coll=GUIDE&dl=GUIDE&CFID=91857573&CFTOKEN=53726080>,(May 2005), pp. 246-251.
Brandt, Raimund "Dynamic Adaptation of Mobile Code in Heterogenous Environments", Der Teschnischen Universitat Munchen,(Feb. 15, 2001), 76 pages.
Brazier, F.M.T. et al., "Generative Migration of Agents", Department of Computer Science, Faculty of Sciences, Vrije Universiteit Amsterdam; de Boelelaan 1081 a, 1081 HV Amsterdam, The Netherlands {frances, bjo, steen, niek} @cs.vu.l,(Aug. 2002), 4 pages.
Christopoulou, Eleni et al., "An ontology-based context management and reasoning process for UbiComp applications", Research Academie Computer Technology Institute, Research Unit 3, Design of Ambient Information Systems Group, N. Kazantzaki str., Rio Campus, 26500, Patras, Greece {hristope, goumop, kameas} @cti.qr,(Oct. 2005), 6 pages.
Hasegawa, Tetsou et al., "Inoperability for mobile agents by incarnation agents", AAMAS; Melbourne, Australia; ACM, 1-58113-683, pp. 1006-1007,(Jul. 14, 2003), 2 pages.
Jih, Wan-Rong et al., "Context-aware Access Control in Pervasive Healthcare", Computer Science and Information Engineering, National Taiwan University, Taiwan jih@agents.csie.ntu.edu.tw., {r93070, yjhsu} @csie.ntu.edu.tw,(2005), 8 pages.
Karnik, Neeran M., et al., "Security in the Ajanta Mobile Agent System", Softw. Pract. Exper. 31, 4 (Apr. 2001), 301-329,(2001), 28 pages.
Korba, Larry "Towards Secure Agent Distribution and Communication", In Proceedings of the 32nd Annual Hawaii International Conference on System Science—vol. 8 (Jan. 5-8, 1999). HICSS. IEEE Computer Society, Washington, DC, 8059.,(1999), 10 pages.
Pauline, Siu Po Lam "Context-Aware State Management for Supporting Mobility in a Pervasive Environment", A thesis submitted to the University of Hong Kong in fulfillment of the thesis requirement for the degree of Master of Philosophy.,(Aug. 2004), 106 pages.
Poggi, Agostino et al., "Security and Trust in Agent-Oriented Middleware", OTM Workshops 2003: 989-1003,(2003), 15 pages.
Qui-Sheng, He et al., "A Lightweight Architecture to Support Context-Aware Ubiquitous Agent System", Department of Computer Science and Engineering, Fudan University, Shanghai 200433, Chain {hequisheng, situ} @fudan.edu.cn,(Aug. 2006), 6 pages.
Walsh, Tom et al., "Security and Reliability in Concordia", In Mobility: Processes, Computers, and Agents ACM Press/Addison-Wesley Publishing Co., New York, NY, 524-534.,(1999), 10 pages.
Yu, Ping et al., "Mobile Agent Enabled Application Mobility for Pervasive Computing", Internet and Mobile Computing Lab, Department of Computing, Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong {cspyu, csjcao, cswen} @comp.polyu.edu.hk.,(OS/2006), 10 pages.
Non Final Office Action, U.S. Appl. No. 10/975,623, (Nov. 12, 2009), 8 pages.
Blackwell, John "Ramit—Rule-Based Alert Management Information Tool", Thesis Florida State University College of Arts and Sciences (2004), pp. 1-69.
FORTE, "Feature Comparison of Agent and Free Agent", Retrieved from: <www.forteinc.com/agent/features.php> on Jan. 26, 2010 (2002),3 pages.
Lockemann, Peter C., "Agents and Databases: Friends or Foes?", Proceedings of the 9th International Database Engineering & Application Symposium (IDEAS '05), (2005), 11 pages.
Russell, Stuart et al., "Artificial intelligence: A Modern Approach", Pearson, (2003), pp. 5, 32-56, 449-454.
Stallings, William "ISDN and Broadband ISDN with Framw Relay and ATM", Prentice-Hall, (1999), p. 516.
Final Office Action, U.S. Appl. No. 11/086,102 (Oct. 20, 2009), 13 pages.
Final Office Action, U.S. Appl. No. 10/975,827, (Nov. 23, 2009), 17 pages.
Final Office Action, U.S. Appl. No. 10/975,146 (Dec. 1, 2009), 11 pages.
Non Final Office Action, U.S. Appl. No. 11/086,120 (Aug. 13, 2009),35 pages.
Advisory Action, U.S. Appl. No. 11/086,121, (Sep. 11, 2009), 3 pages.
Notice of Allowance, U.S. Appl. No. 11/645,190 (Sep. 17, 2009),8 pages.
Notice of Allowance, U.S. Appl. No. 11/645,193, (Sep. 17, 2009), 8 pages.
Advisory Action, U.S. Appl. No. 11/086,121 (Sep. 11, 2009),3 pages.
Final Office Action, U.S. Appl. No. 11/086,211, (Sep. 29, 2009), 18 pages.
Praset, et al., "Processor Modeling and Code Selection for Retargetable Compilation", Jul. 2001 ACM TODAES vol. 6 Issue 3 (Jul. 2001), pp. 277-307.
Notice of Allowance, U.S. Appl. No. 11/645,291 (Oct. 1, 2009), 11 pages.
Final Office Action, U.S. Appl. No. 11/052,680, (Oct. 5, 2009), 16 pages.
Non Final Office Action, U.S. Appl. No. 11/086,102, (Jun. 2, 2009), 18 pages.
Non Final Office Action, U.S. Appl. No. 11/645,306, (Jun. 4, 2009), 19 pages.
Non Final Office Action, U.S. Appl. No. 10/975,623, (Jun. 9, 2009), 6 pages.
Non Final Office Action, U.S. Appl. No. 11/086,121, (Jun. 22, 2009), 32 pages.
Non Final Office Action, U.S. Appl. No. 11/086,211, 25 pages.
Requirement for Restriction/Election, U.S. Appl. No. 11/086,102, (Apr. 6, 2009), 7 pages.
Non Final Office Action, U.S. Appl. No. 10/975,827, (Apr. 7, 2009), 38 pages.
Final Office Action, U.S. Appl. No. 11/086,120, (Apr. 14, 2009), 48 pages.
Non Final Office Action, U.S. Appl. No. 11/645,292, (Apr. 17, 2009), 40 pages.
Final Office Action, U.S. Appl. No. 11/052,680 (Feb. 13, 2009), 117 pages.
Flanagan, Roxy "Graphic + Internet related tutorials Using Winzip", indexed by www.archive.org,(Oct. 30, 2008),8 pages.
Non Final Office Action, U.S. Appl. No. 11/645,190, (Mar. 13, 2009), 41 pages.
Non Final Office Action, U.S. Appl. No. 11/645,291, (Mar. 13, 2009), 41 pages.
Non Final Office Action, U.S. Appl. No. 11/645,193, (Mar. 16, 2009), 41 pages.
Notice of Allowance, U.S. Appl. No. 11/086,101, (Mar. 23, 2009),20 pages.

(56) **References Cited**

OTHER PUBLICATIONS

Non Final Office Action, U.S. Appl. No. 11/052,680, (May 6, 2009),28 pages.
Non Final Office Action, U.S. Appl. No. 11/645,300, (May 11, 2009), 21 pages.
Non Final Office Action, U.S. Appl. No. 11/645,302, (May 11, 2009), 21 pages.
"Stuffit Deluxe", Aladdin Systems,(Jan. 1, 2004), 83 pages.
Gu, T. et al., "Toward an OSGi-based infrastructure for context-aware applications", U.S. Appl. No. 11/645,190, Pervasive Computing IEEE, vol. 3, Issue 4. Digital abject Identifier 10.11 09/MPRV.2004. 19,(Oct.-Dec. 2004), 9 pages.
Pantic, M et al., "Simple agent framework: an educational tool introducing the basics of AI programming", U.S. Appl. No. 11/645,190, Information Technology: Research and Education, 2003. Proceedings. ITRE2003.,(Aug. 11-13, 2003), 426-430.
Singh, V.K. et al., "DYSWIS: An architecture for automated diagnosis of networks", U.S. Appl. No. 11/645,190, Operations and Management Symposium, 2008. NOMS 2008. IEEE,(Apr. 7-11, 2008), 851-854.
Non Final Office Action, U.S. Appl. No. 10/975,146, (May 26, 2009), 21 pages.
Emmerich, et al., "TIGRA—An Architectural Style for Enterprise Application Integration", IEEE, (Jul. 2001), p. 567-576.
Bussler, Christoph "The Role of B2B Engines in B2B Integration Architectures", ACM, (Mar. 2002), 67-72.
Jandl, et al., "Enterprise Application Integration by means of a generic CORBA LDAP Gateway", ACM, (May 2002), 711.
Sutherland, et al., "Enterprise Application Integration and Complex Adaptive Systems", ACM, (Oct. 2002), 59-64.
Ballance, et al., "The Pan Language-Based Editing System for Integrated Development Environments", ACM, (Oct. 1990), 77-93.
Dilts, et al., "An Inteligent Interface to CIM Multiple Data Bases", ACM, TALIP, (Mar. 2004),vol. 3 Issue 1, pp. 491-509.
Shiao, Dan, "Mobile Agent: New Model of Intelligent Distributed Computing Part One" Oct. 2004.
Final Office Action on U.S. Appl. No. 11/645,198, mailed Nov. 8, 2011.
Non-Final Office Action on U.S. Appl. No. 11/645,198, mailed Jul. 22, 2011.

* cited by examiner

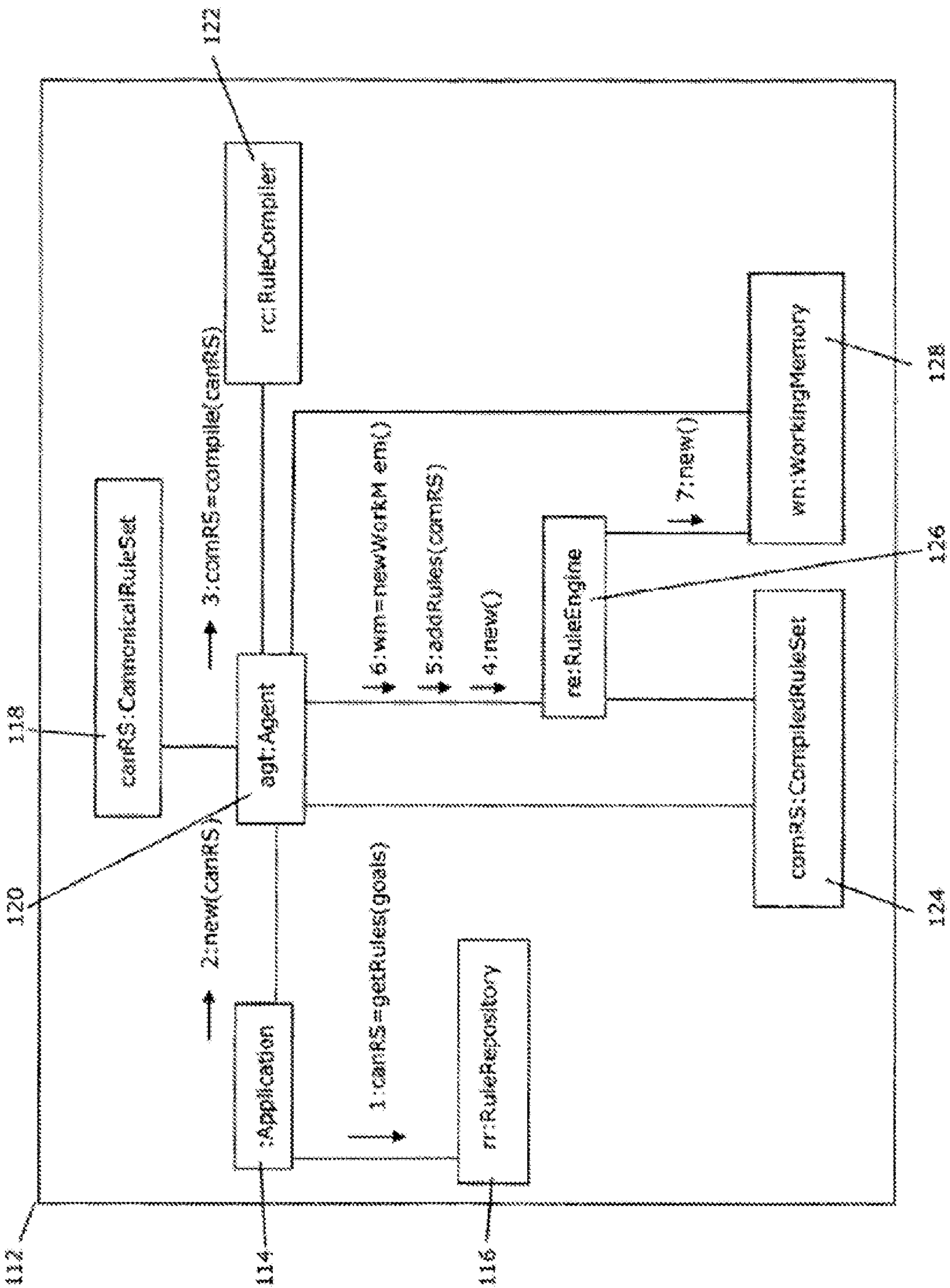

122
rc:RuleCompiler
118
canRS:CanonicalRuleSet
3:comRS=compile(canRS)
120
agt:Agent
6:wm=newWorkMem()
5:addRules(comRS)
4:new()
re:RuleEngine
126
7:new()
wm:WorkingMemory
128
comRS:CompiledRuleSet
124
2:new(canRS)
:Application
1:canRS=getRules(goals)
rr:RuleRepository
112
114
116

Figure 1

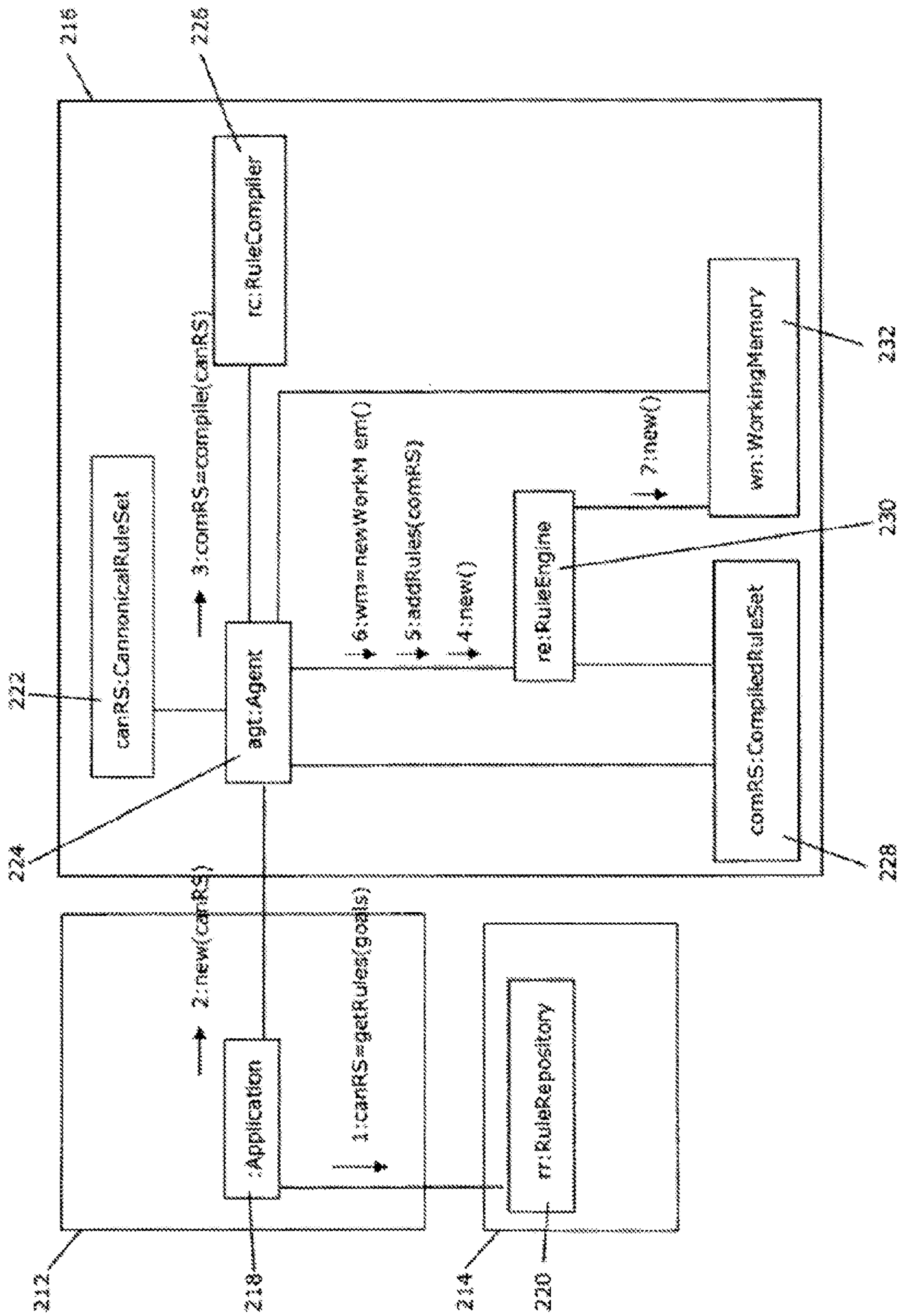

216
226
rc:RuleCompiler
canRS:CannonicalRuleSet
3:comRS=compile(canRS)
222
agt:Agent
224
6:wm=newWorkM em()
5:addRules(comRS)
4:new()
re:RuleEngine
7:new()
wm:WorkingMemory
232
230
comRS:CompiledRuleSet
228
2:new(canRS)
:Application
1:canRS=getRules(goals)
rr:RuleRepository
212
218
214
220

Figure 2

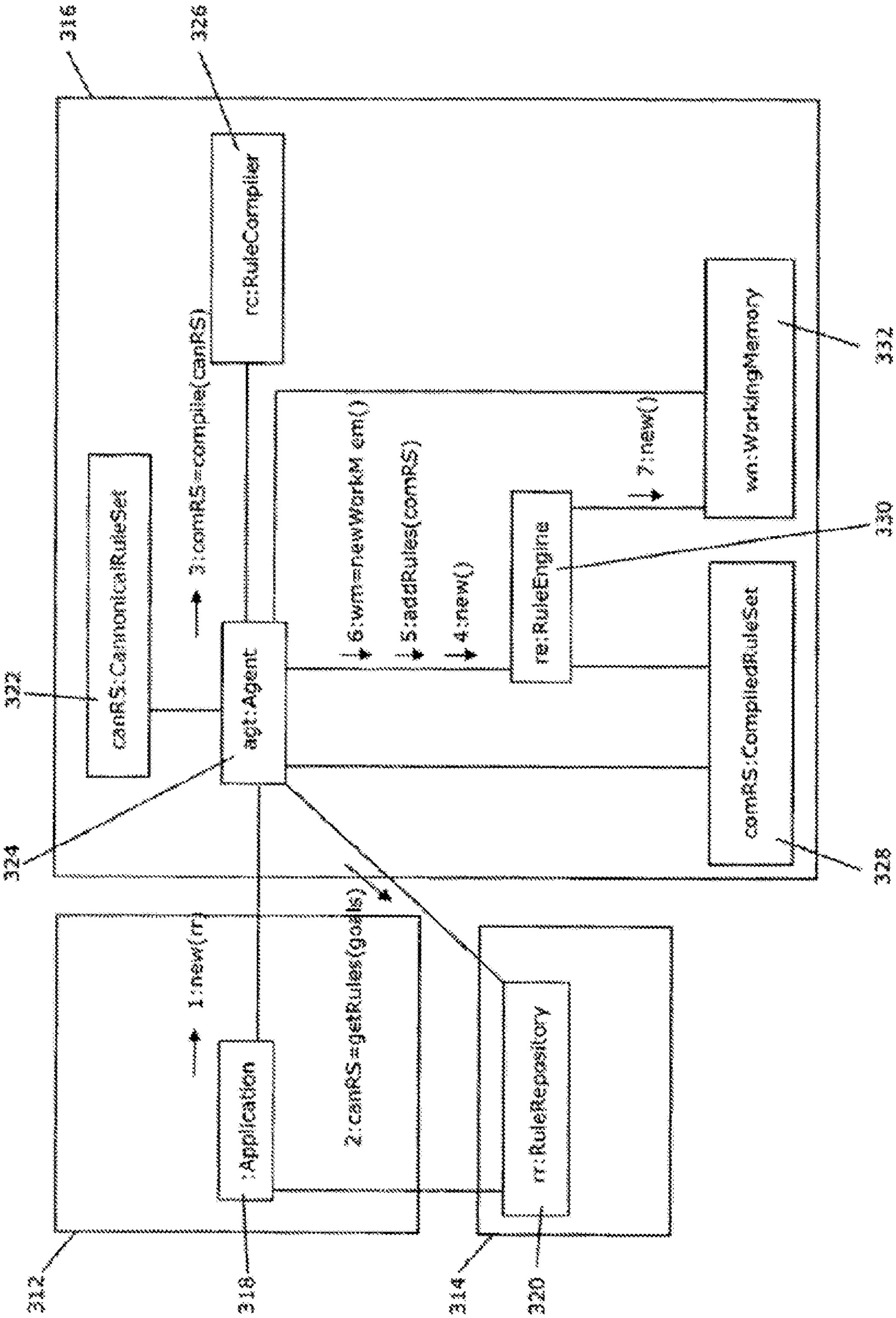

316
326
rc:RuleCompiler
3:comRS=compile(canRS)
canRS:CanonicalRuleSet
322
act:Agent
324
6:wm=newWorkMem()
5:addRules(comRS)
4:new()
re:RuleEngine
330
7:new()
wm:WorkingMemory
332
comRS:CompiledRuleSet
328
1:new(rr)
:Application
2:canRS=getRules(goals)
rr:RuleRepository
312
318
314
320

Figure 3

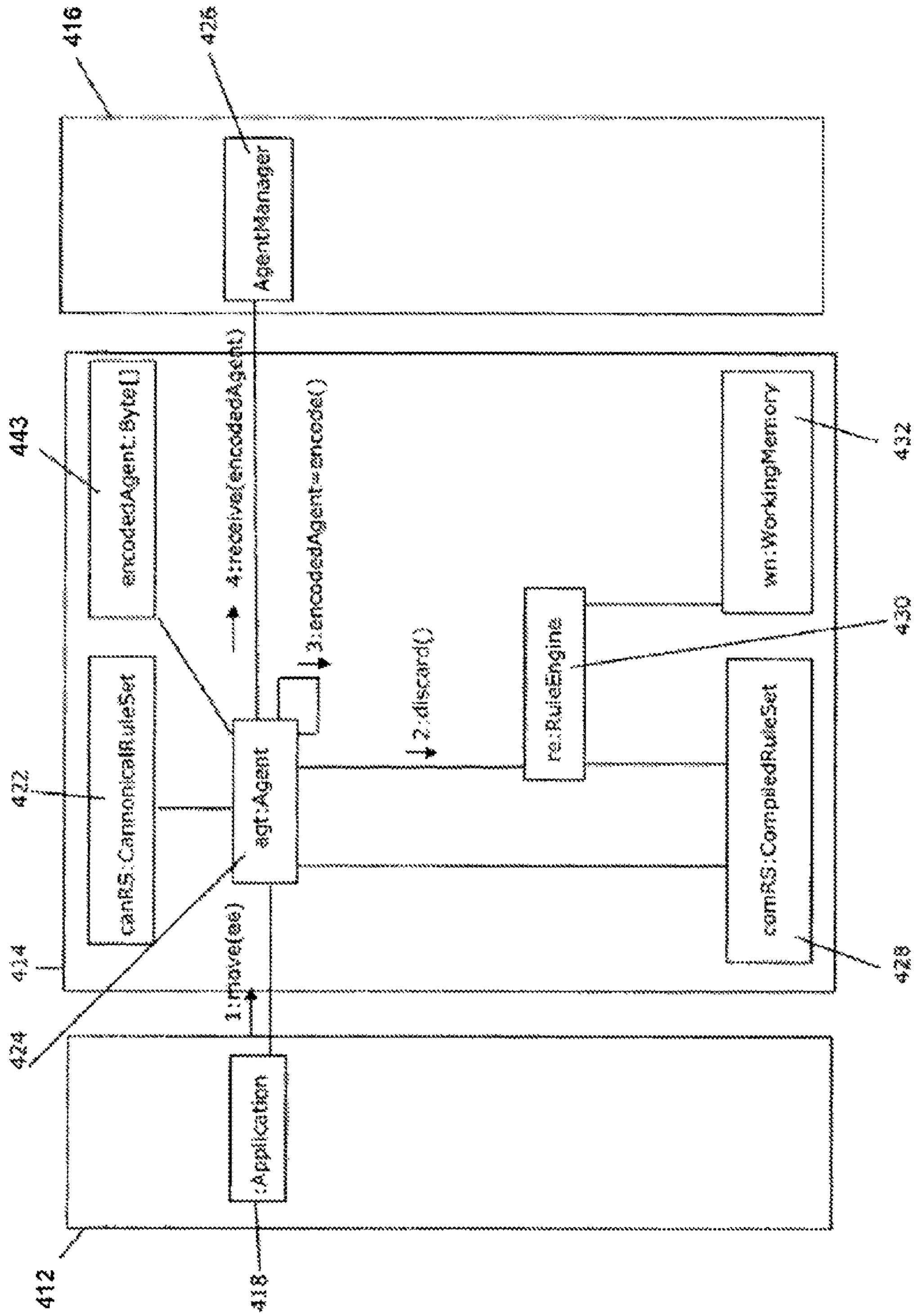

416
426
AgentManager
443
encodedAgent:Byte[]
4:receive(encodedAgent)
3:encodedAgent=encode()
2:discard()
422
canRS:CanonicalRuleSet
agt:Agent
re:RuleEngine
wm:WorkingMemory
432
430
comRS:CompiledRuleSet
428
434
1:move(ee)
424
:Application
412
418

Figure 4

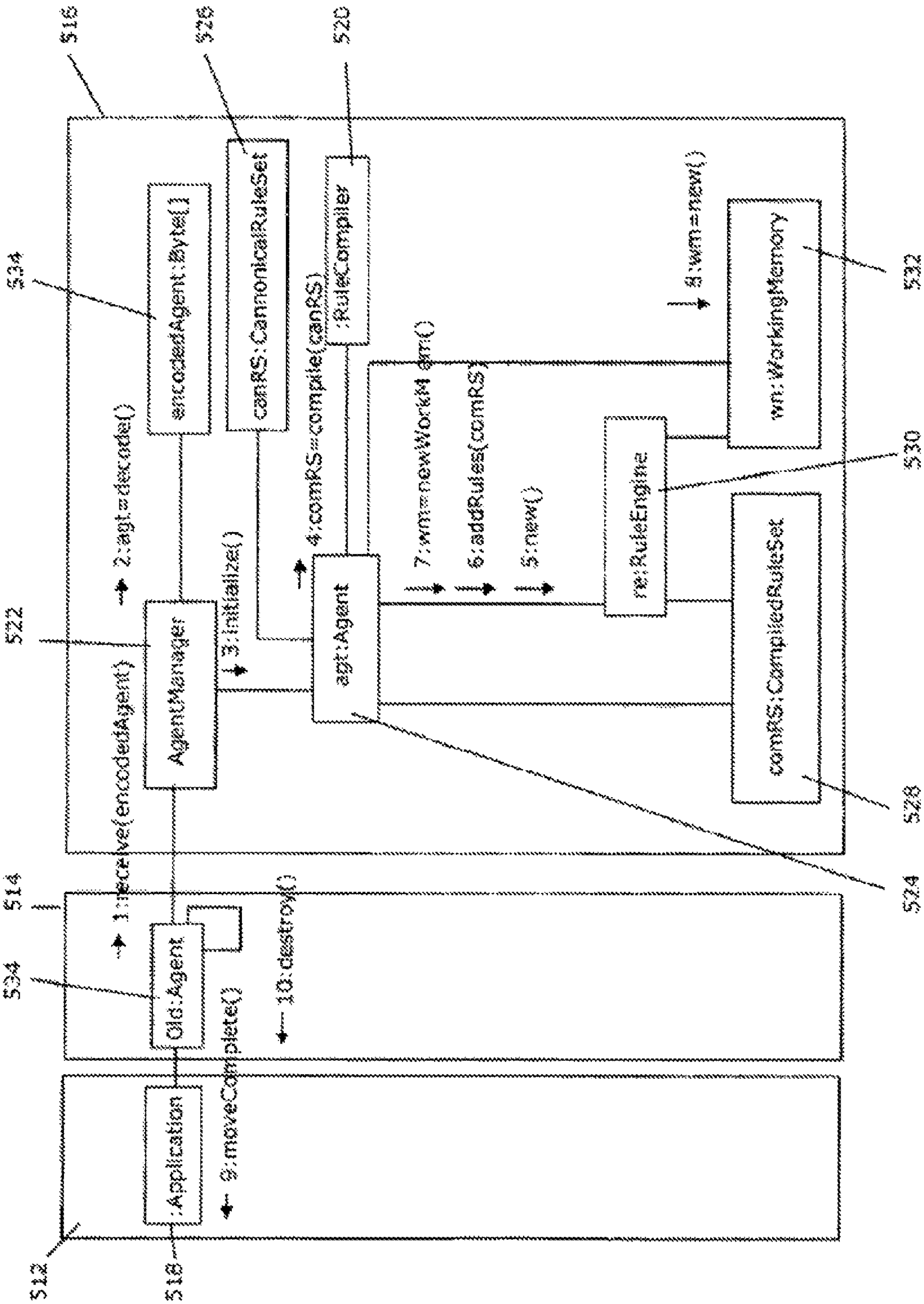

512
514
516
518
520
522
524
526
528
530
532
534
534
:Application
Old:Agent
AgentManager
agt:Agent
encodedAgent:Byte[]
canRS:CannonicalRuleSet
:RuleCompiler
re:RuleEngine
wm:WorkingMemory
comRS:CompiledRuleSet
1:receive(encodedAgent)
2:agt=decode()
3:initialize()
4:comRS=compile(canRS)
5:new()
6:addRules(comRS)
7:wm=newWorkMem()
8:wm=new()
9:moveComplete()
10:destroy()

Figure 5

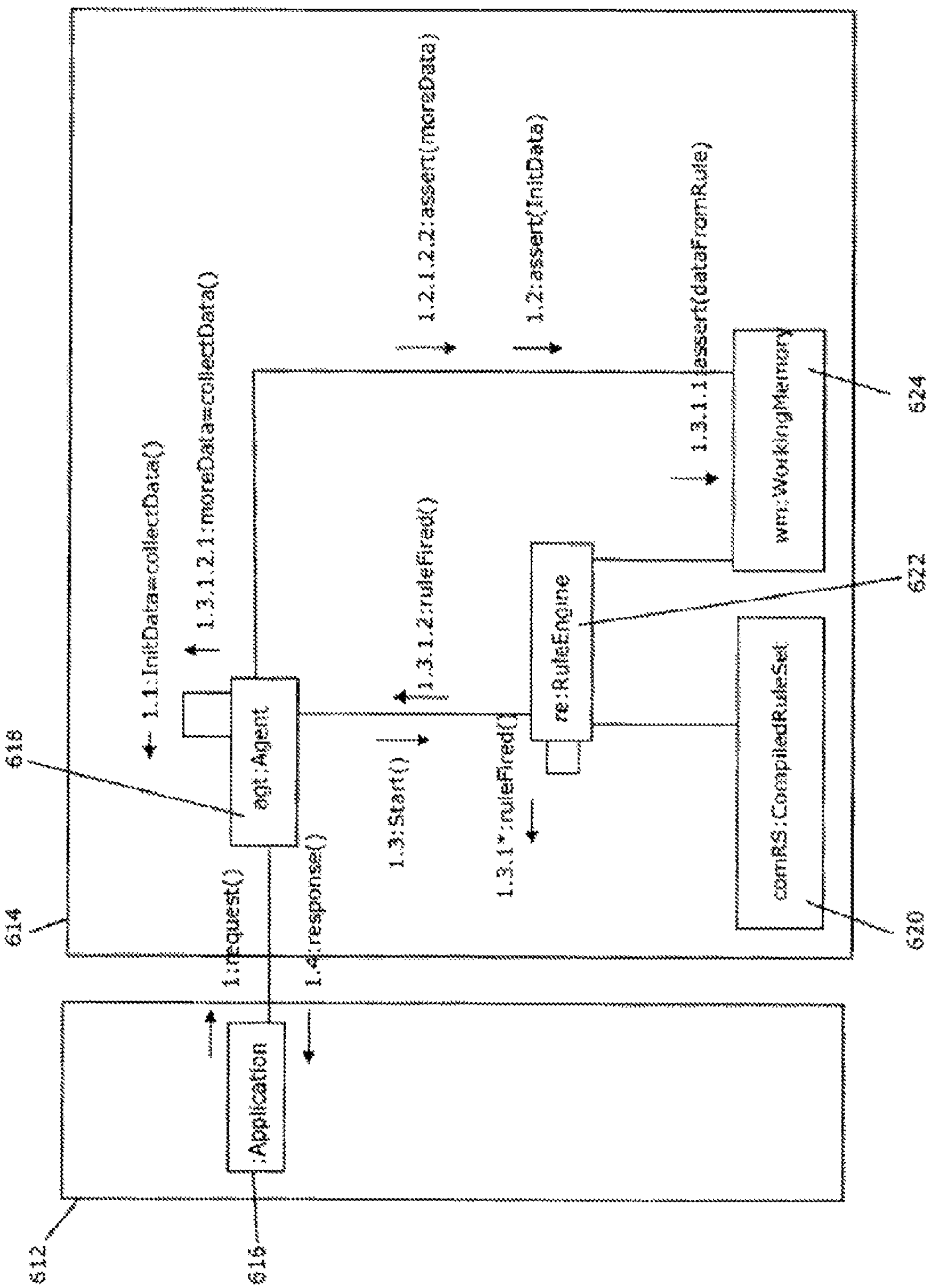

612
614
616
618
620
622
624
:Application
agt:Agent
re:RulesEngine
comRS:CompiledRuleSet
wm:WorkingMemory
1:request()
1.4:response()
1.1:InitData=collectData()
1.3.1.2.1:moreData=collectData()
1.2.1.2.2:assert(moreData)
1.2:assert(InitData)
1.3.1.1:assert(dataFromRule)
1.3.1.2:ruleFired()
1.3:Start()
1.3.1*:ruleFired()

Figure 6

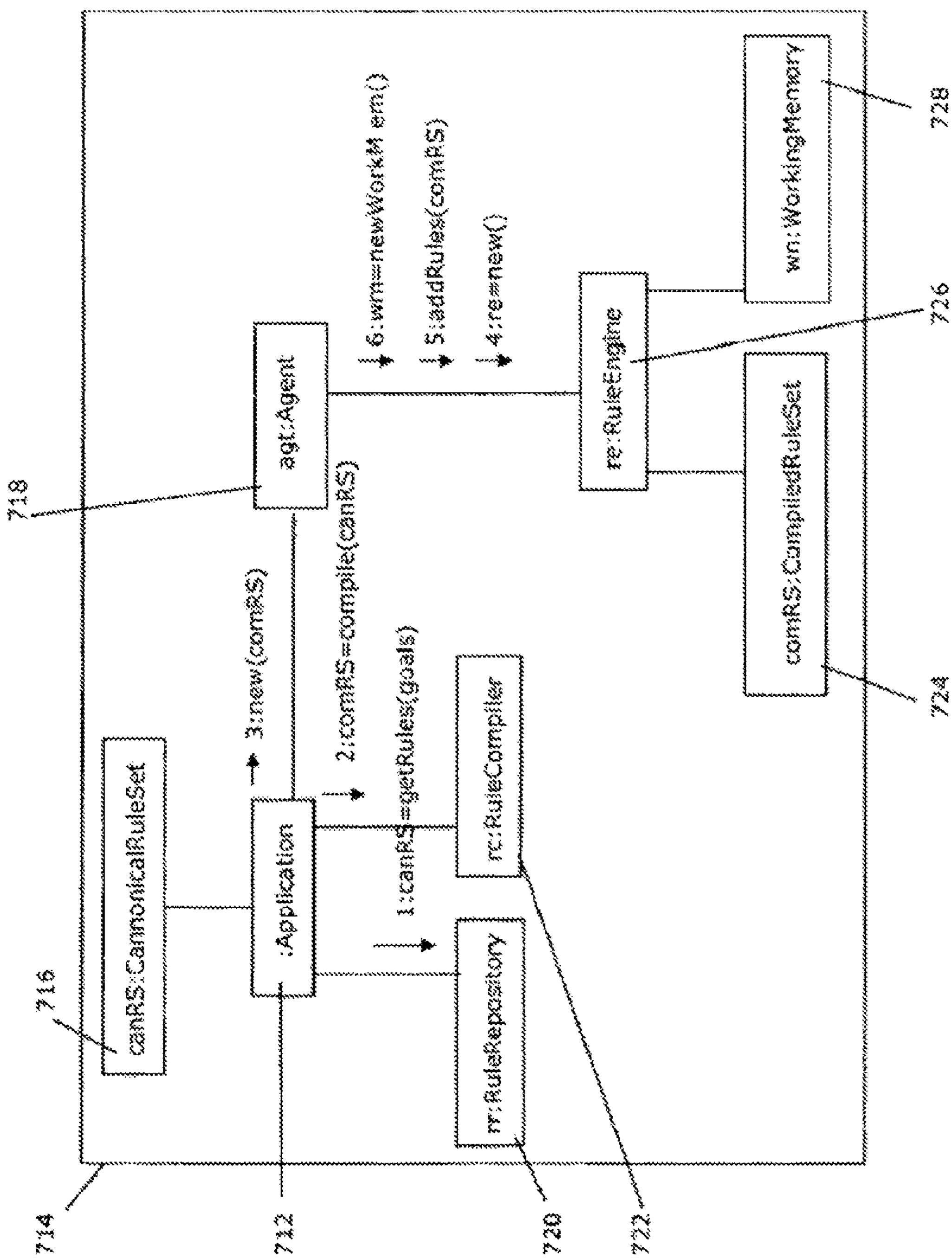
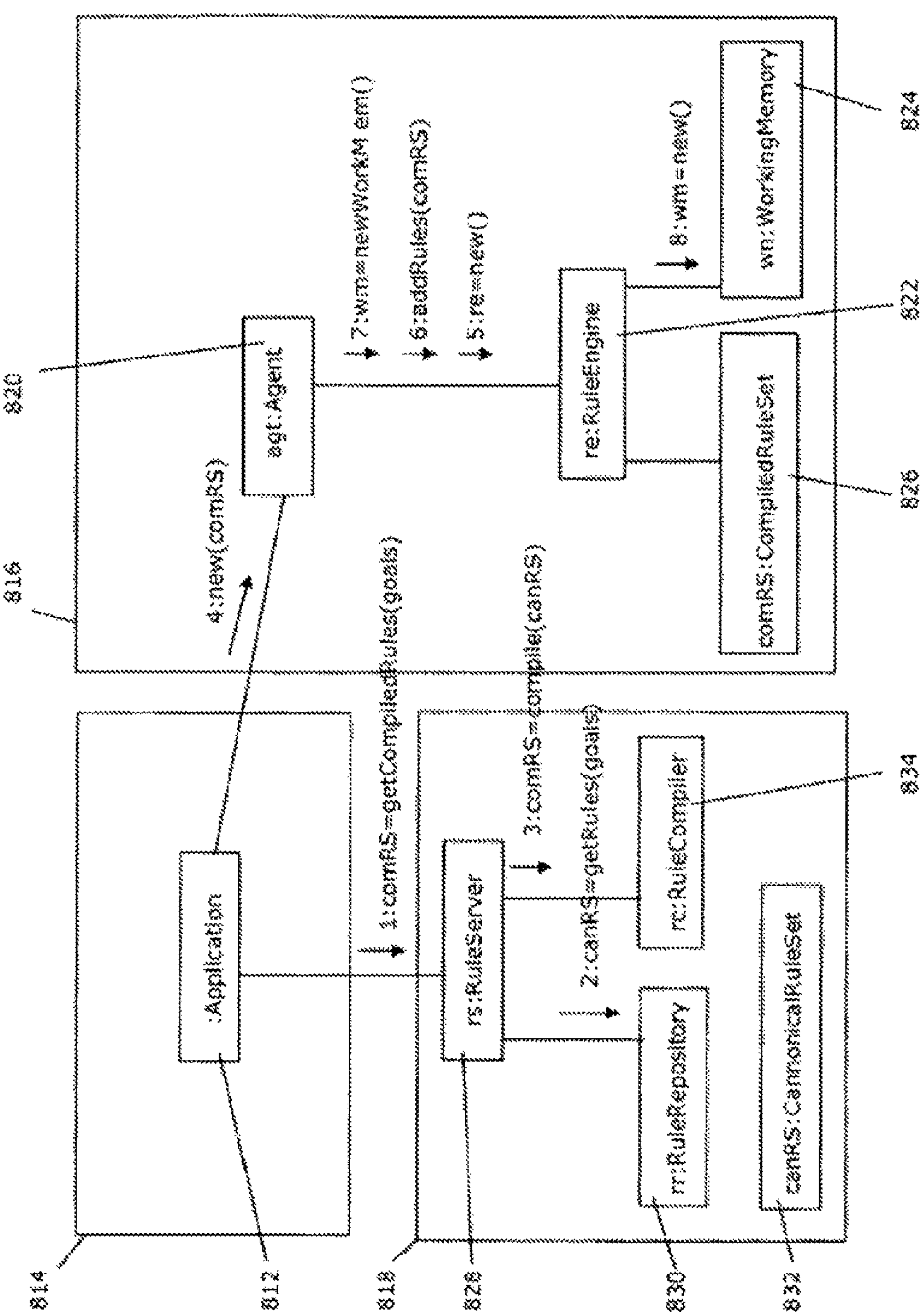

714
712
720
722
716
718
canRS:CanonicalRuleSet
:Application
agt:Agent
rr:RuleRepository
rc:RuleCompiler
1:canRS=getRules(goals)
2:comRS=compile(canRS)
3:new(comRS)
4:re=new()
5:addRules(comRS)
6:wm=newWorkMem()
re:RuleEngine
comRS:CompiledRuleSet
wm:WorkingMemory
724
726
728

Figure 7



814
812
818
828
830
832
816
820
:Application
rs:RuleServer
rr:RuleRepository
rc:RuleCompiler
canRS:CanonicalRuleSet
1:comRS=getCompiledRules(goals)
2:canRS=getRules(goals)
3:comRS=compile(canRS)
4:new(comRS)
agt:Agent
re:RuleEngine
comRS:CompiledRuleSet
wm:WorkingMemory
5:re=new()
6:addRules(comRS)
7:wm=newWorkMem()
8:wm=new()
822
824
826
834

Figure 8

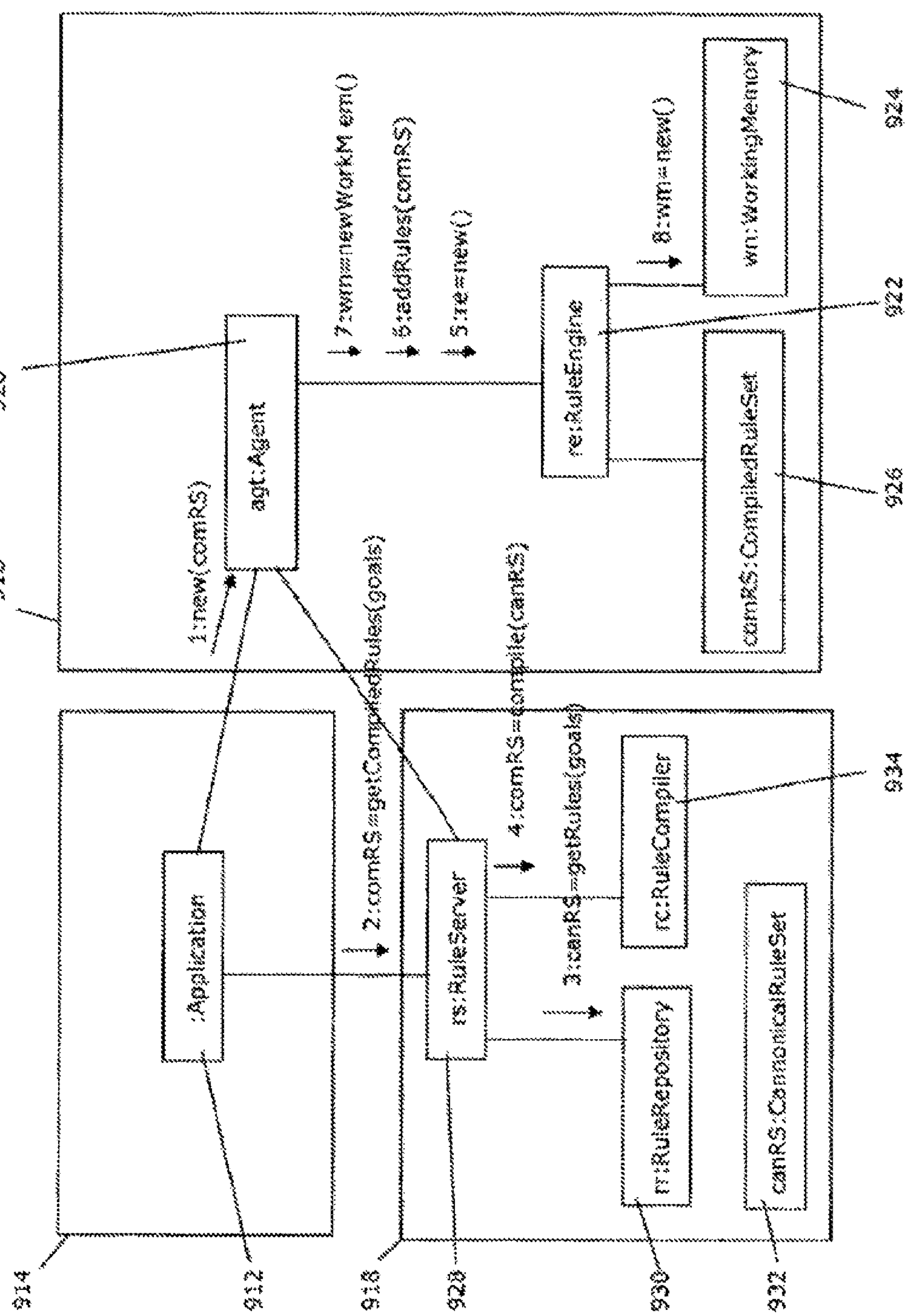
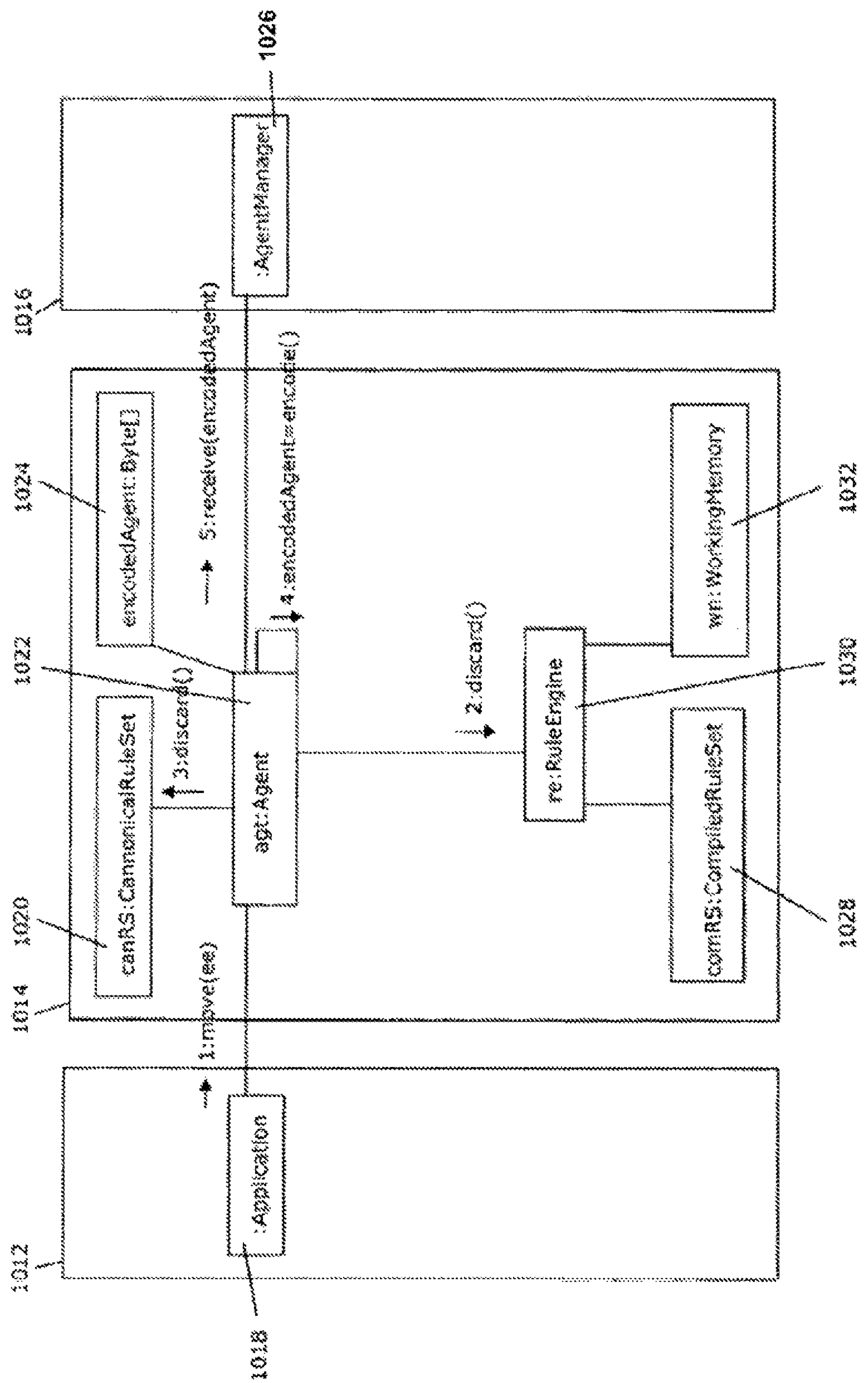

916
920
agt:Agent
1:new(comRS)
7:wm=newWorkMem()
6:addRules(comRS)
5:re=new()
re:RuleEngine
8:wm=new()
wm:WorkingMemory
924
922
comRS:CompiledRuleSet
926
914
:Application
912
918
2:comRS=getCompiledRules(goals)
rs:RuleServer
928
4:comRS=compile(canRS)
3:canRS=getRules(goals)
rc:RuleCompiler
934
rr:RuleRepository
930
canRS:CannonicalRuleSet
932

Figure 9



1026
1016
:AgentManager
5:receive(encodedAgent)
4:encodedAgent=encode()
1024
encodedAgent:Byte[]
1032
wm:WorkingMemory
2:discard()
1022
3:discard()
agt:Agent
re:RuleEngine
1030
1020
canRS:CanonicalRuleSet
comRS:CompiledRuleSet
1028
1014
1:move(ee)
:Application
1012
1018

Figure 10

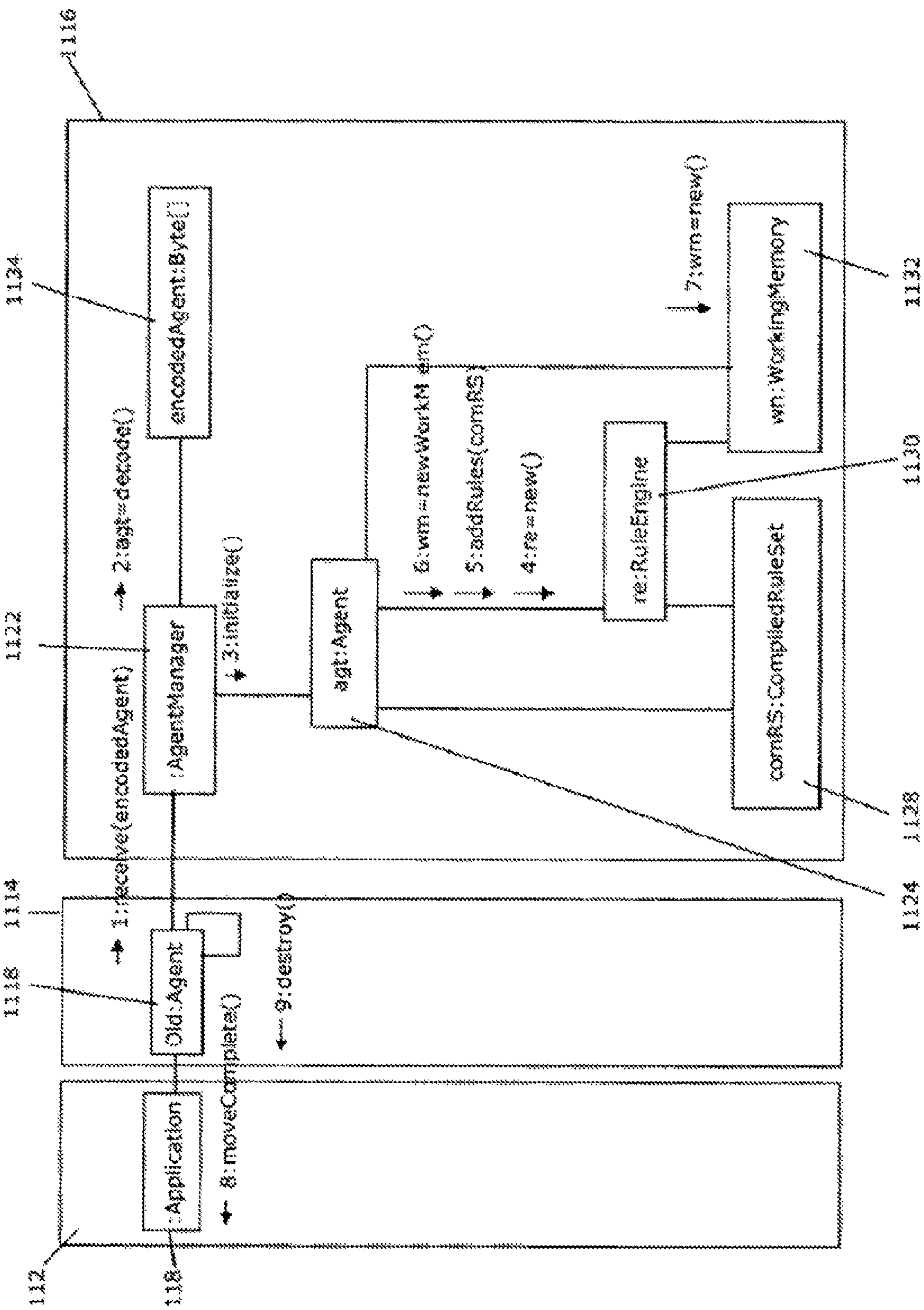

1116
1124
encodedAgent:Byte[]
1122
:AgentManager
2:agt=decode()
1:receive(encodedAgent)
3:initialize()
agt:Agent
6:wm=newWorkMem()
5:addRules(comRS)
4:re=new()
re:RuleEngine
wm:WorkingMemory
7:wm=new()
comRS:CompiledRuleSet
1132
1130
1128
1124
1114
1118
Old:Agent
9:destroy()
8:moveComplete()
:Application
1112
1118

Figure 11

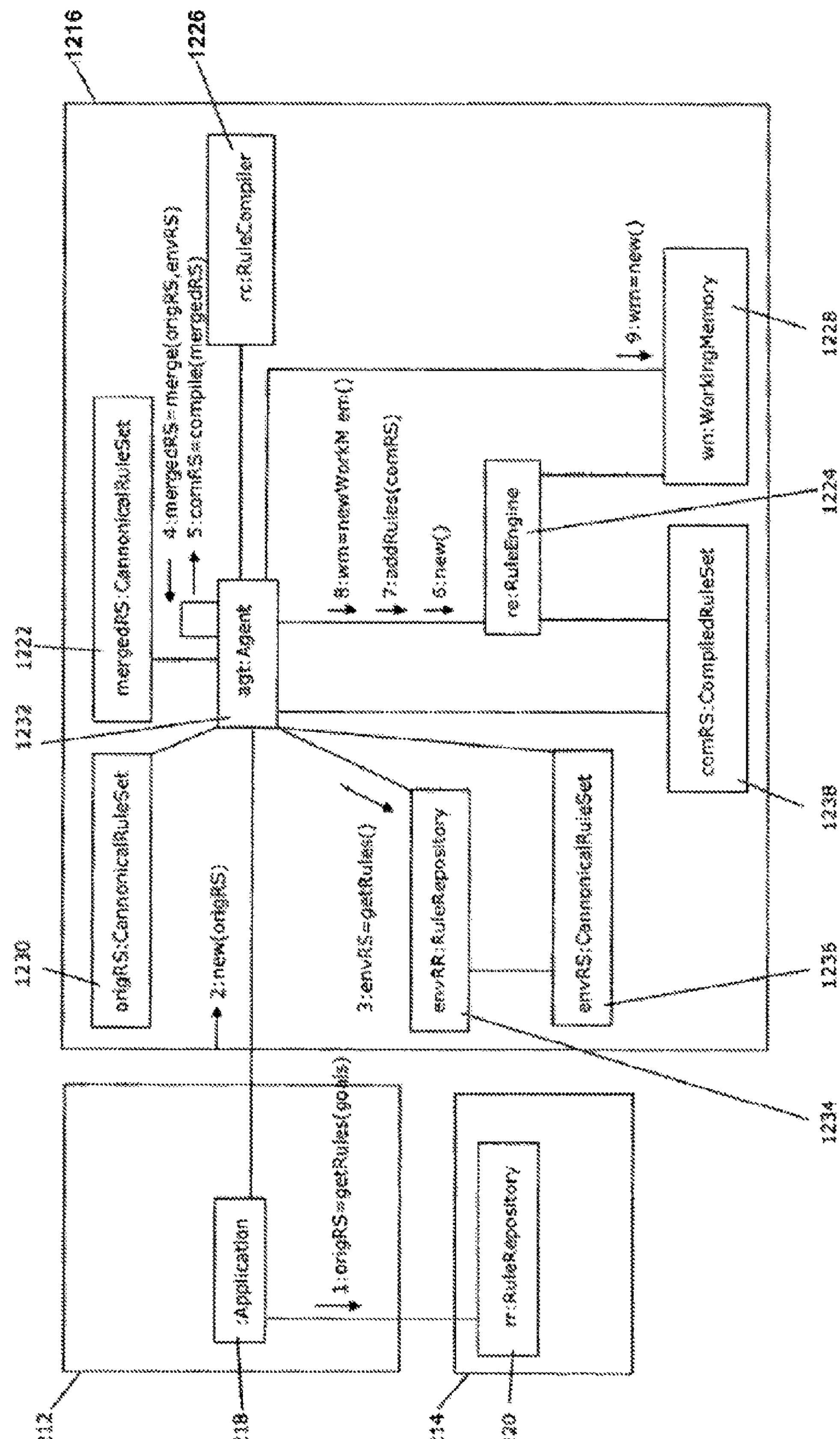

1216
1226
rc:RuleCompiler
mergedRS:CannonicalRuleSet
4:mergedRS=merge(origRS,envRS)
5:comRS=compile(mergedRS)
agt:Agent
8:wm=newWorkM em()
7:addRules(comRS)
6:new()
re:RuleEngine
9:wm=new()
wm:WorkingMemory
comRS:CompiledRuleSet
1222
1232
1224
1228
origRS:CannonicalRuleSet
2:new(origRS)
3:envRS=getRules()
envRR:RuleRepository
envRS:CannonicalRuleSet
1230
1236
1234
1238
:Application
1:origRS=getRules(goals)
rr:RuleRepository
1212
1214
1220

Figure 12

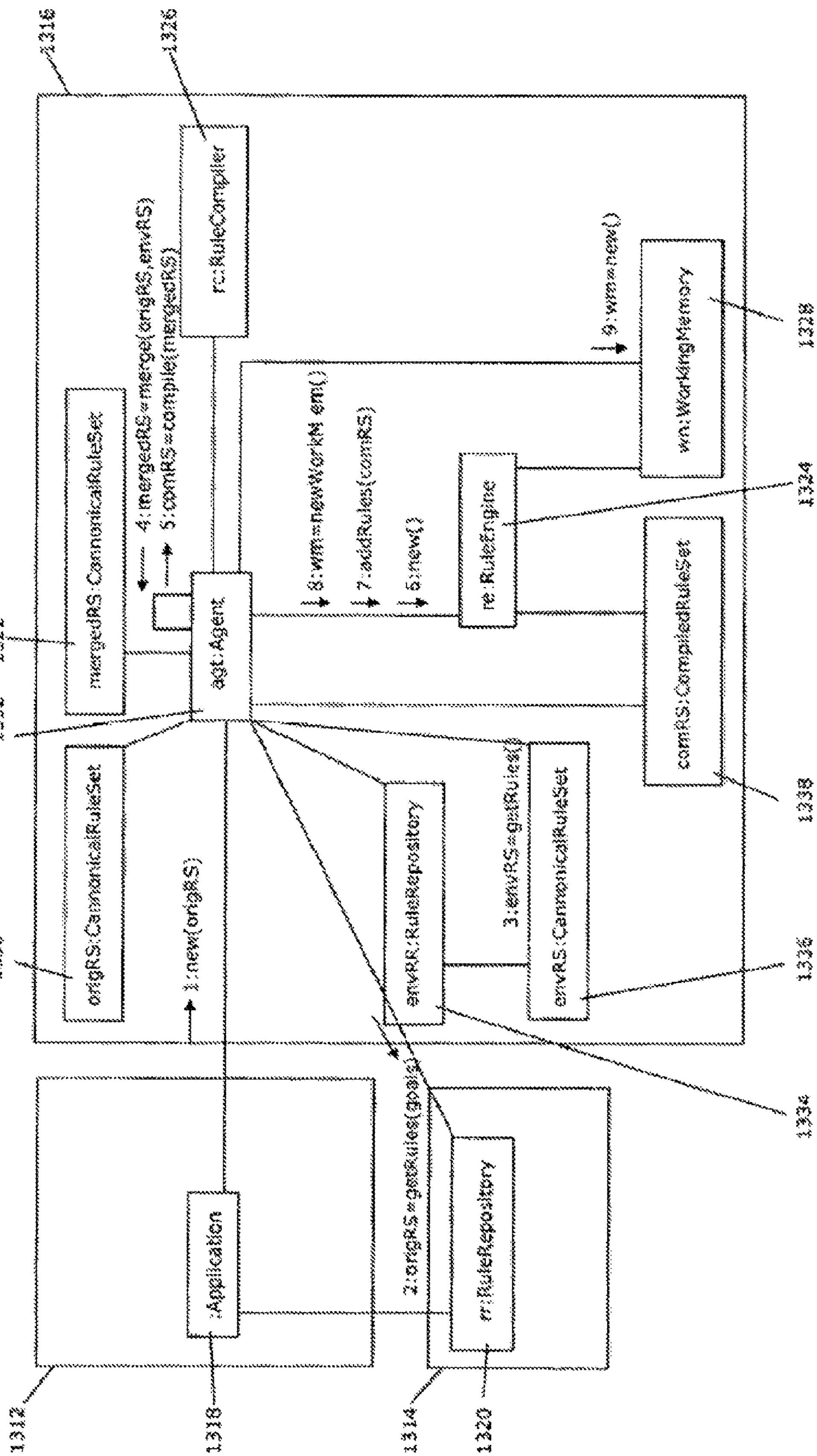

1316
1326
rc:RuleCompiler
4:mergedRS=merge(origRS,envRS)
5:comRS=compile(mergedRS)
9:wm=new()
wm:WorkingMemory
1328
1322
mergedRS:CanonicalRuleSet
8:wm=newWorkingMem()
7:addRules(comRS)
6:new()
re:RuleEngine
1324
1332
agt:Agent
comRS:CompiledRuleSet
1338
1330
origRS:CanonicalRuleSet
1:new(origRS)
envRR:RuleRepository
3:envRS=getRules()
envRS:CanonicalRuleSet
1336
1334
:Application
2:origRS=getRules(goals)
rr:RuleRepository
1312
1318
1314
1320

Figure 13

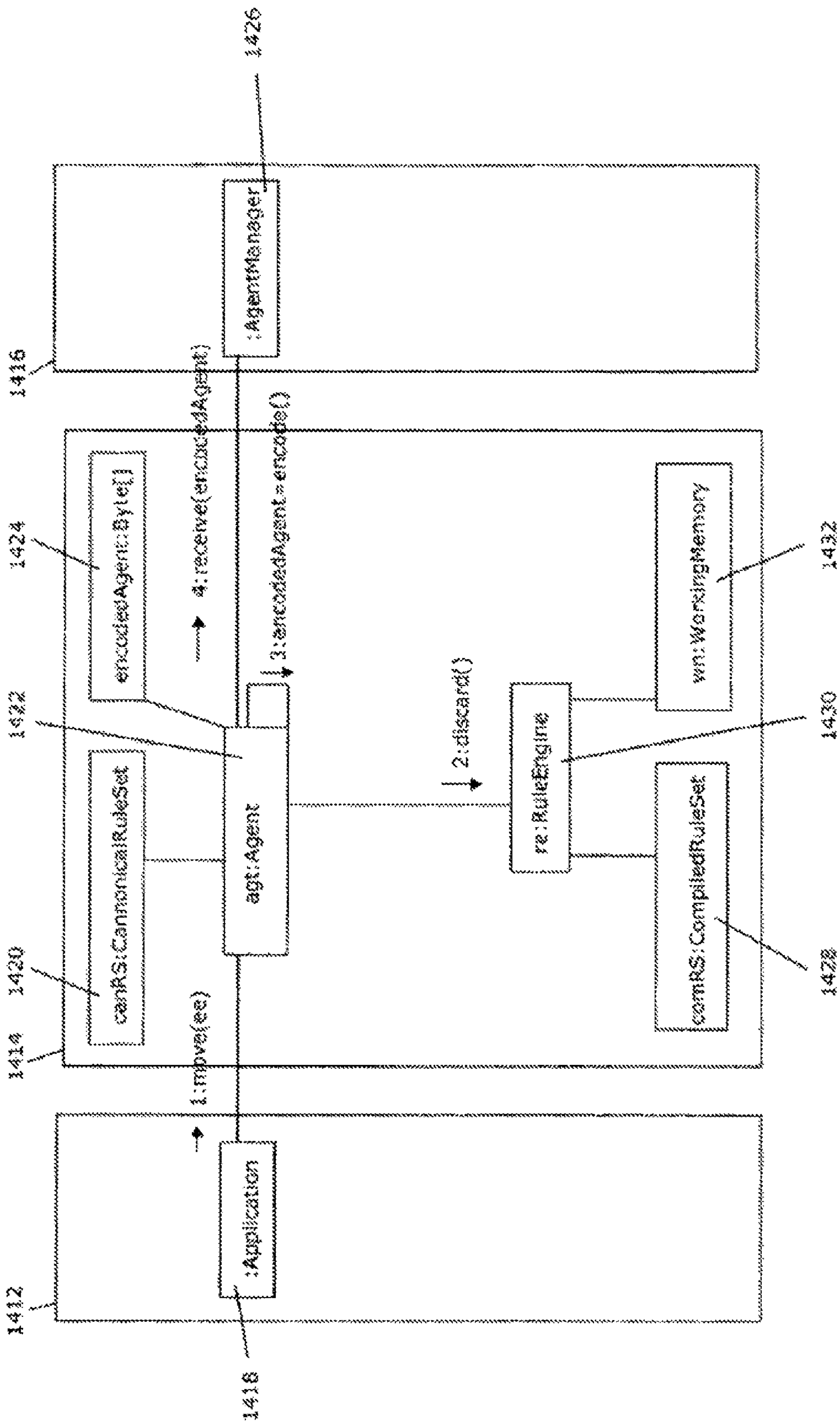

1412
1414
1416
1418
1420
1422
1424
1426
1428
1430
1432
:Application
1:move(ee)
canRS:CannonicalRuleSet
agt:Agent
encodedAgent:Byte[]
4:receive(encodedAgent)
3:encodedAgent=encode()
:AgentManager
2:discard()
re:RuleEngine
comRS:CompiledRuleSet
wm:WorkingMemory

Figure 14

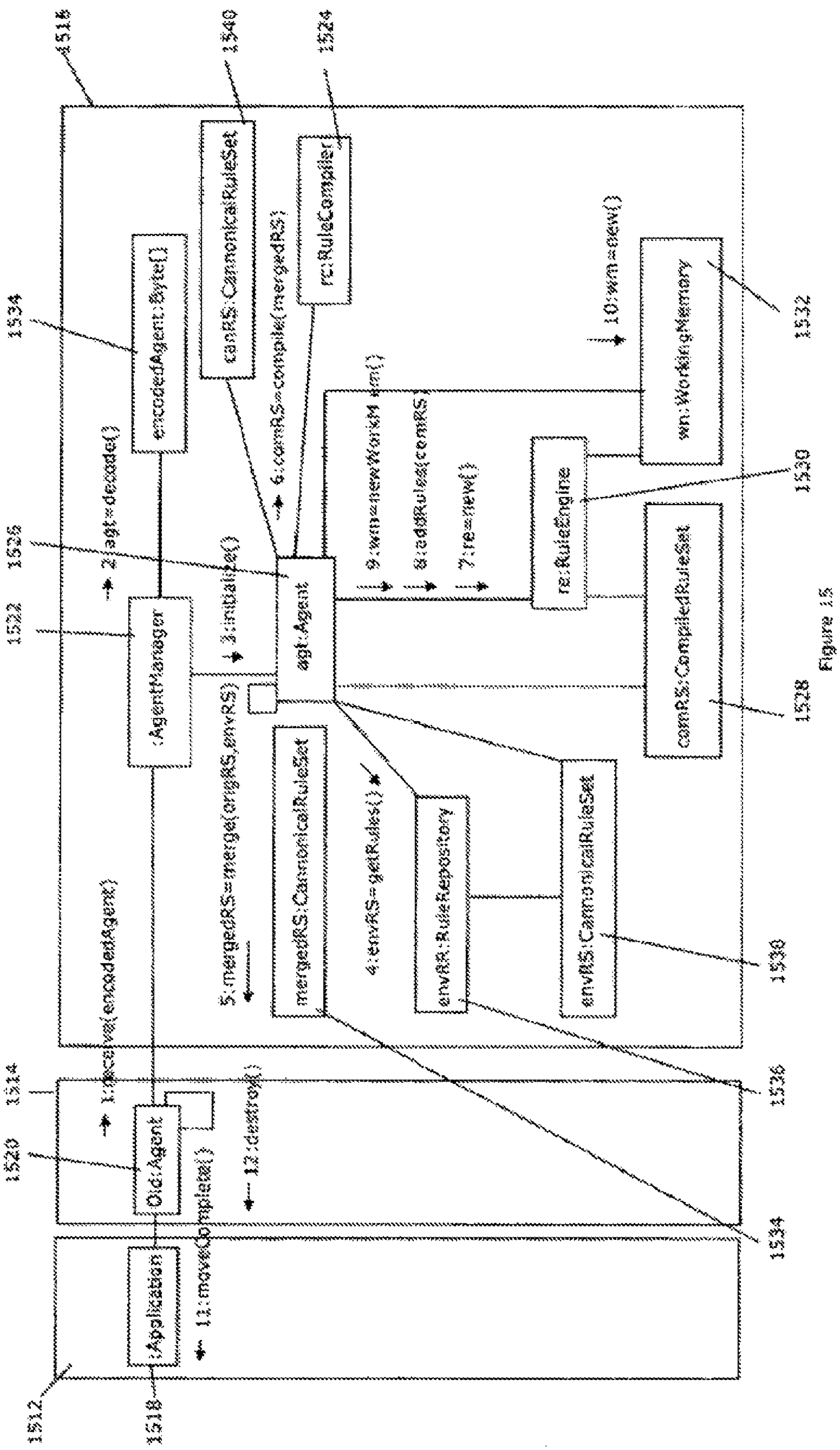

1512
1514
1516
1518
1520
1522
1524
1526
1528
1530
1532
1534
1536
1540
:Application
11:moveComplete()
Old:Agent
1:receive(encodedAgent)
12:destroy()
:AgentManager
2:agt=decode()
encodedAgent:Byte[]
3:initialize()
agt:Agent
comRS:CannonicalRuleSet
6:comRS=compile(mergedRS)
rc:RuleCompiler
9:wm=newWorkMem()
8:addRules(comRS)
7:re=new()
re:RuleEngine
10:wm=new()
wm:WorkingMemory
comRS:CompiledRuleSet
5:mergedRS=merge(origRS,envRS)
mergedRS:CannonicalRuleSet
4:envRS=getRules()
envRR:RuleRepository
envRS:CannonicalRuleSet

Figure 15

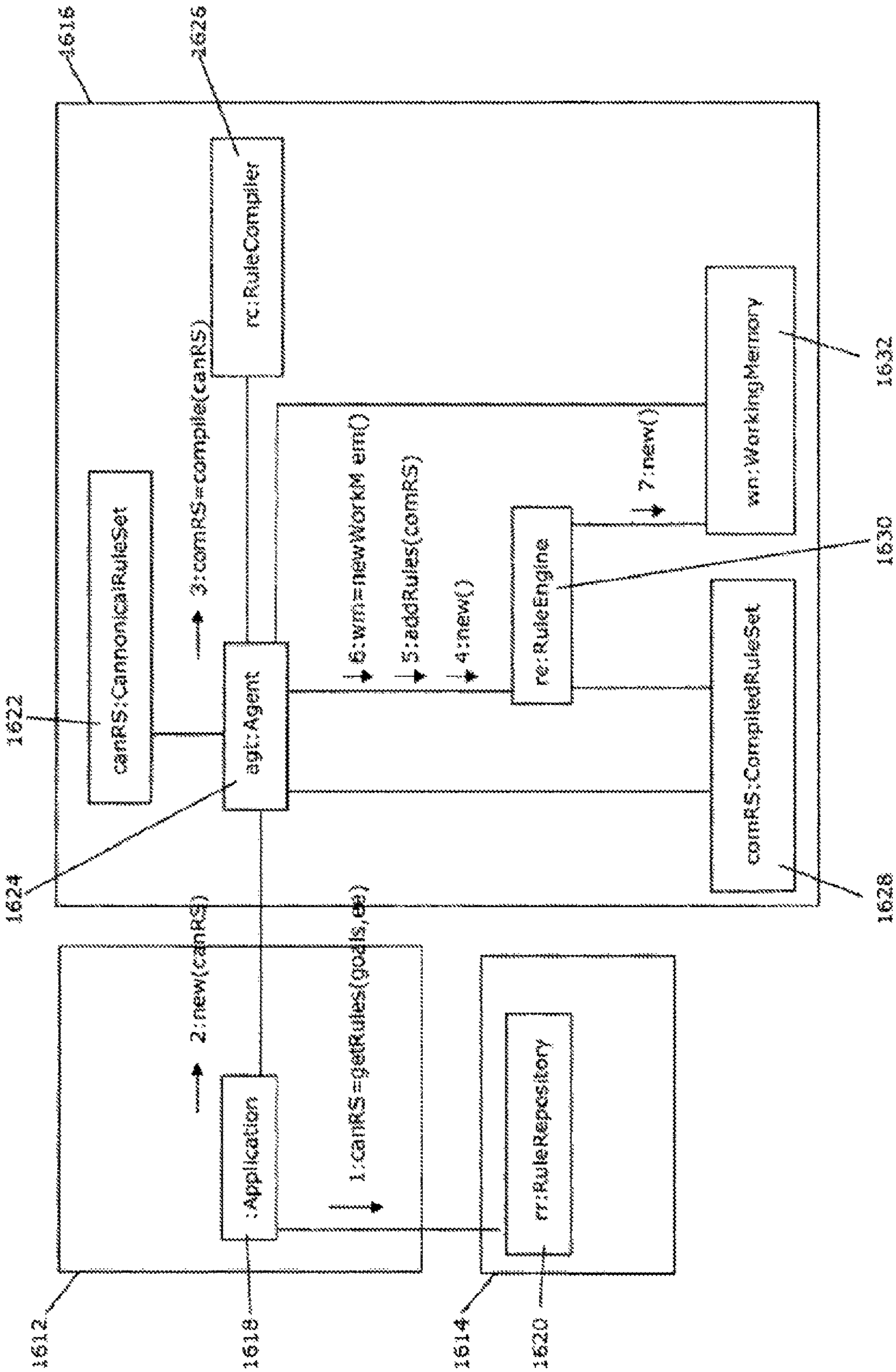

1616
1626
rc:RuleCompiler
1622
canRS:CannonicalRuleSet
3:comRS=compile(canRS)
1624
agt:Agent
6:wm=newWorkM em()
5:addRules(comRS)
4:new()
re:RuleEngine
7:new()
wm:WorkingMemory
1632
1630
comRS:CompiledRuleSet
1628
2:new(canRS)
:Application
1:canRS=getRules(goals,ex)
rr:RuleRepository
1612
1618
1614
1620

Figure 16

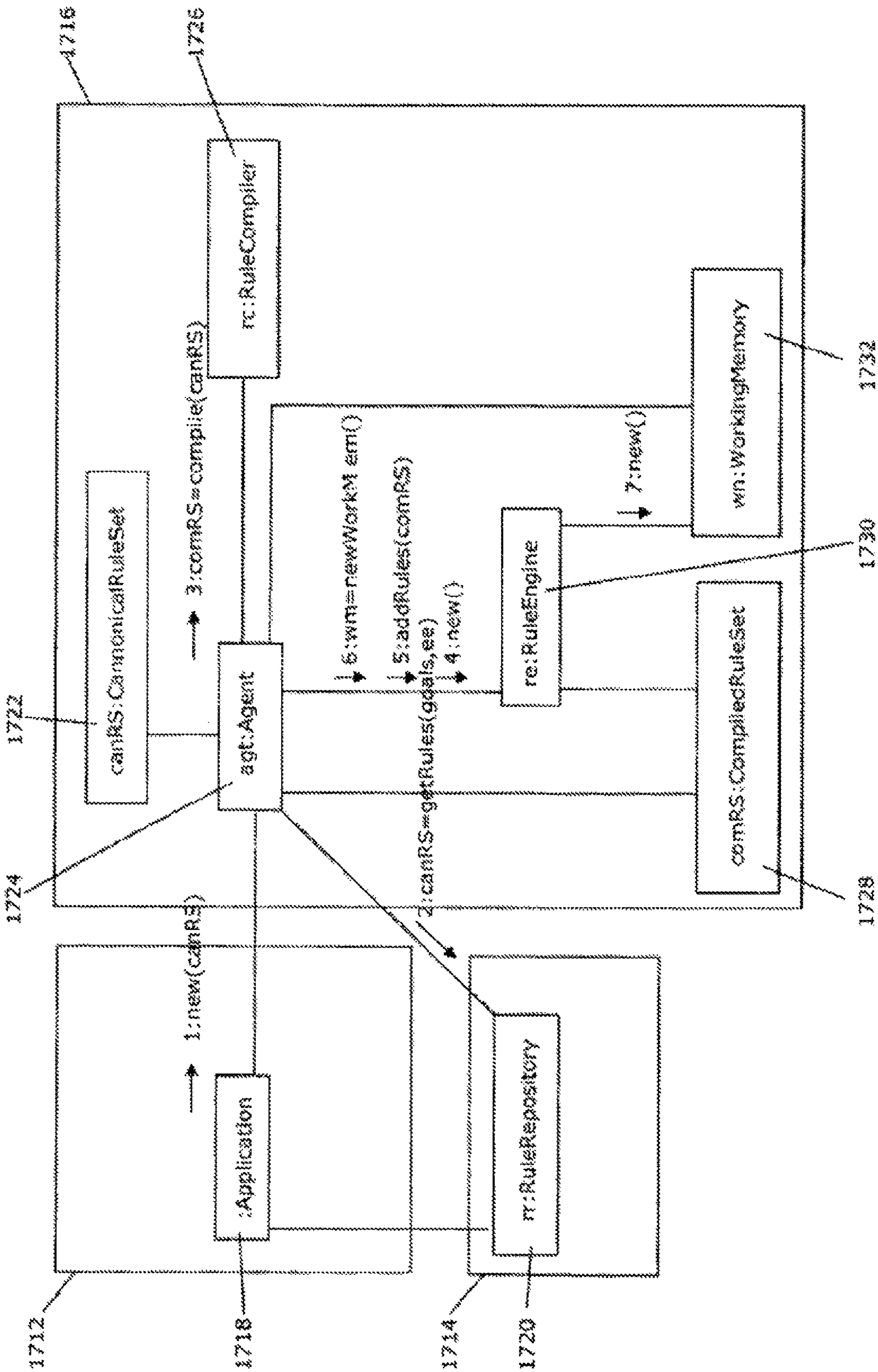

1716
1726
rc:RuleCompiler
3:comRS=compile(canRS)
1722
canRS:CanonicalRuleSet
1724
agt:Agent
6:wm=newWorkM em()
5:addRules(comRS)
4:new()
re:RuleEngine
1730
7:new()
wn:WorkingMemory
1732
comRS:CompiledRuleSet
1728
2:canRS=getRules(goals,ee)
1:new(canRS)
:Application
rr:RuleRepository
1712
1718
1714
1720

Figure 17

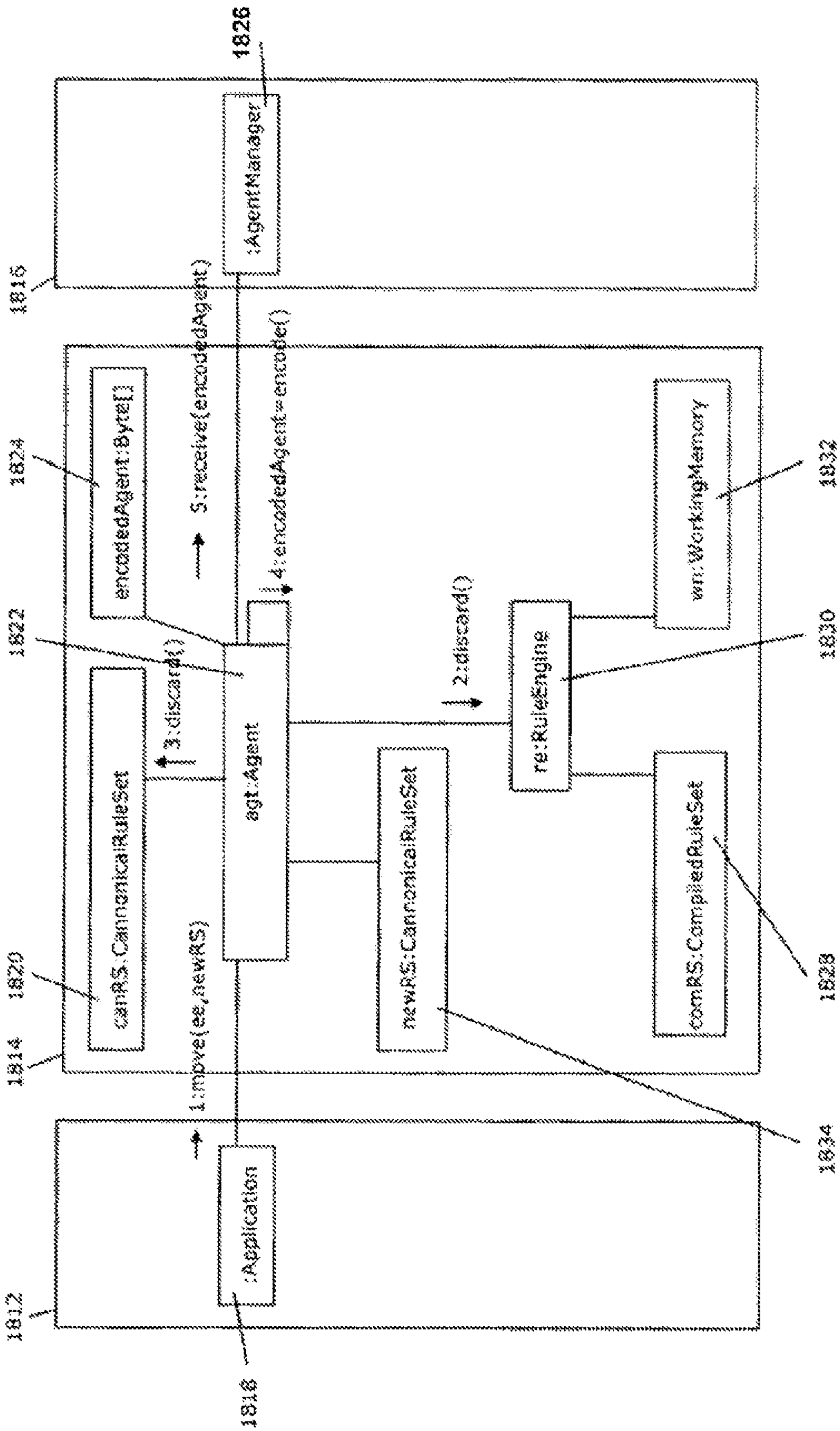

1812
1814
1816
1818
1820
1822
1824
1826
1828
1830
1832
1834
:Application
:AgentManager
1:move(ee,newRS)
agt:Agent
canRS:CanonicalRuleSet
newRS:CanonicalRuleSet
encodedAgent:Byte[]
re:RuleEngine
wm:WorkingMemory
comRS:CompiledRuleSet
2:discard()
3:discard()
4:encodedAgent=encode()
5:receive(encodedAgent)

Figure 18

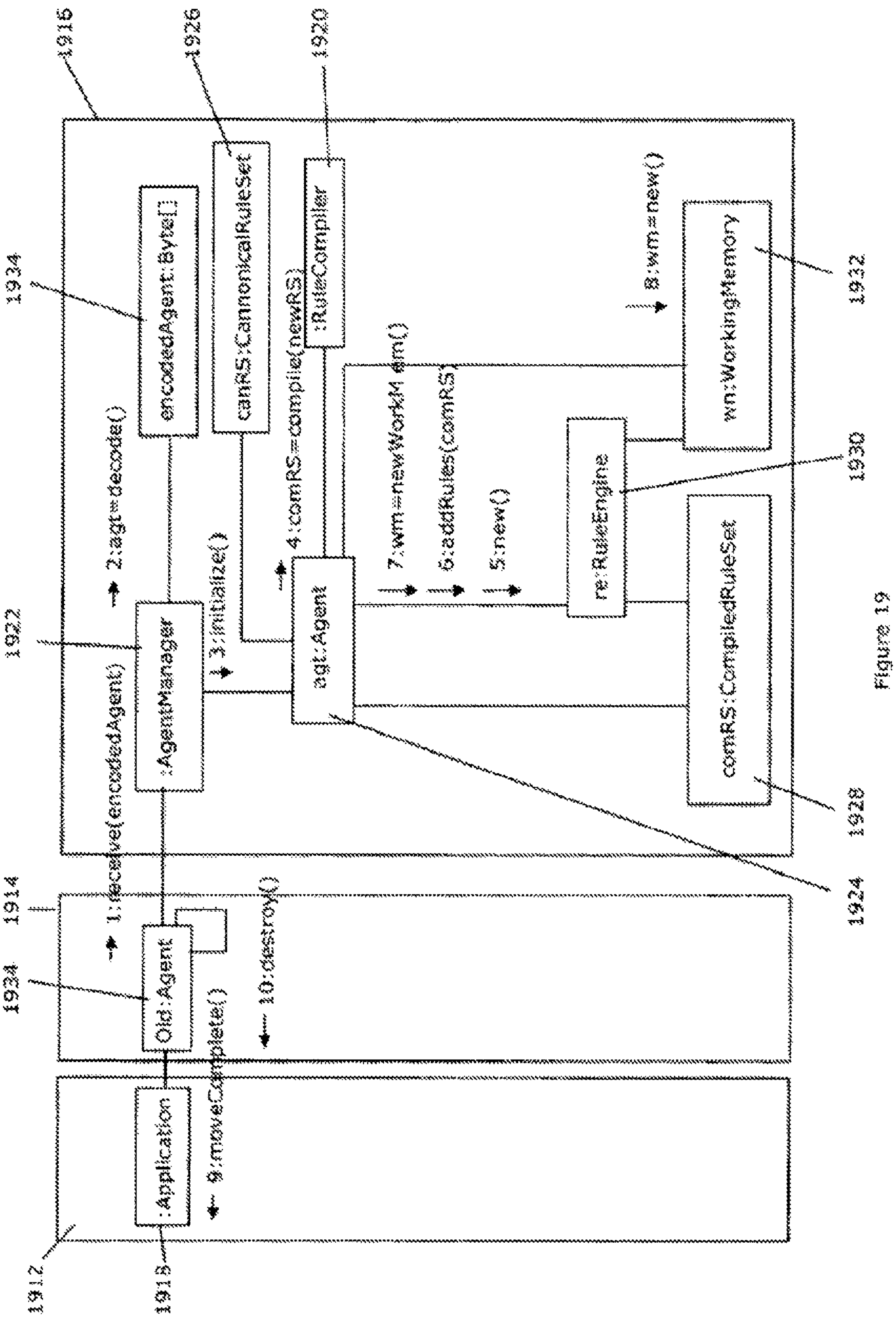
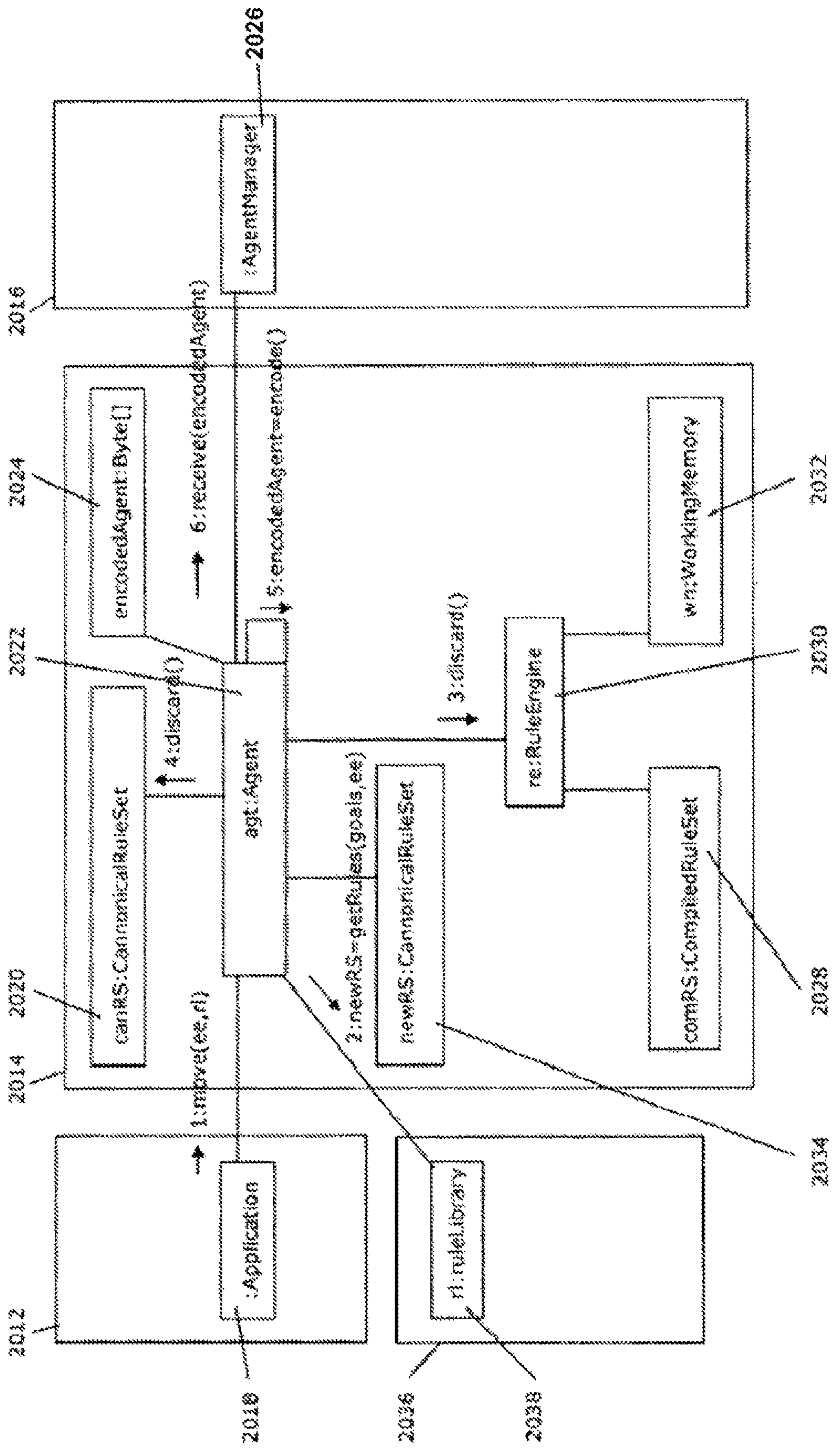

1916
1926
1920
1934
1922
1932
1930
1928
1924
1914
1934
1912
1918
encodedAgent:Byte[]
canRS:CannonicalRuleSet
:RuleCompiler
:AgentManager
agt:Agent
re:RuleEngine
wm:WorkingMemory
comRS:CompiledRuleSet
Old:Agent
:Application
1:receive(encodedAgent)
2:agt=decode()
3:initialize()
4:comRS=compile(newRS)
5:new()
6:addRules(comRS)
7:wm=newWorkMem()
8:wm=new()
9:moveComplete()
10:destroy()

Figure 19



2026
:AgentManager
2016
6:receive(encodedAgent)
5:encodedAgent=encode()
2024
encodedAgent:Byte[]
2022
4:discard()
agt:Agent
2020
canRS:CannonicalRuleSet
2014
1:move(ee,rl)
2:newRS=getRules(goals,ee)
newRS:CannonicalRuleSet
3:discard()
re:RuleEngine
2030
wm:WorkingMemory
2032
comRS:CompiledRuleSet
2028
2034
2012
:Application
2018
rl:ruleLibrary
2036
2038

Figure 20

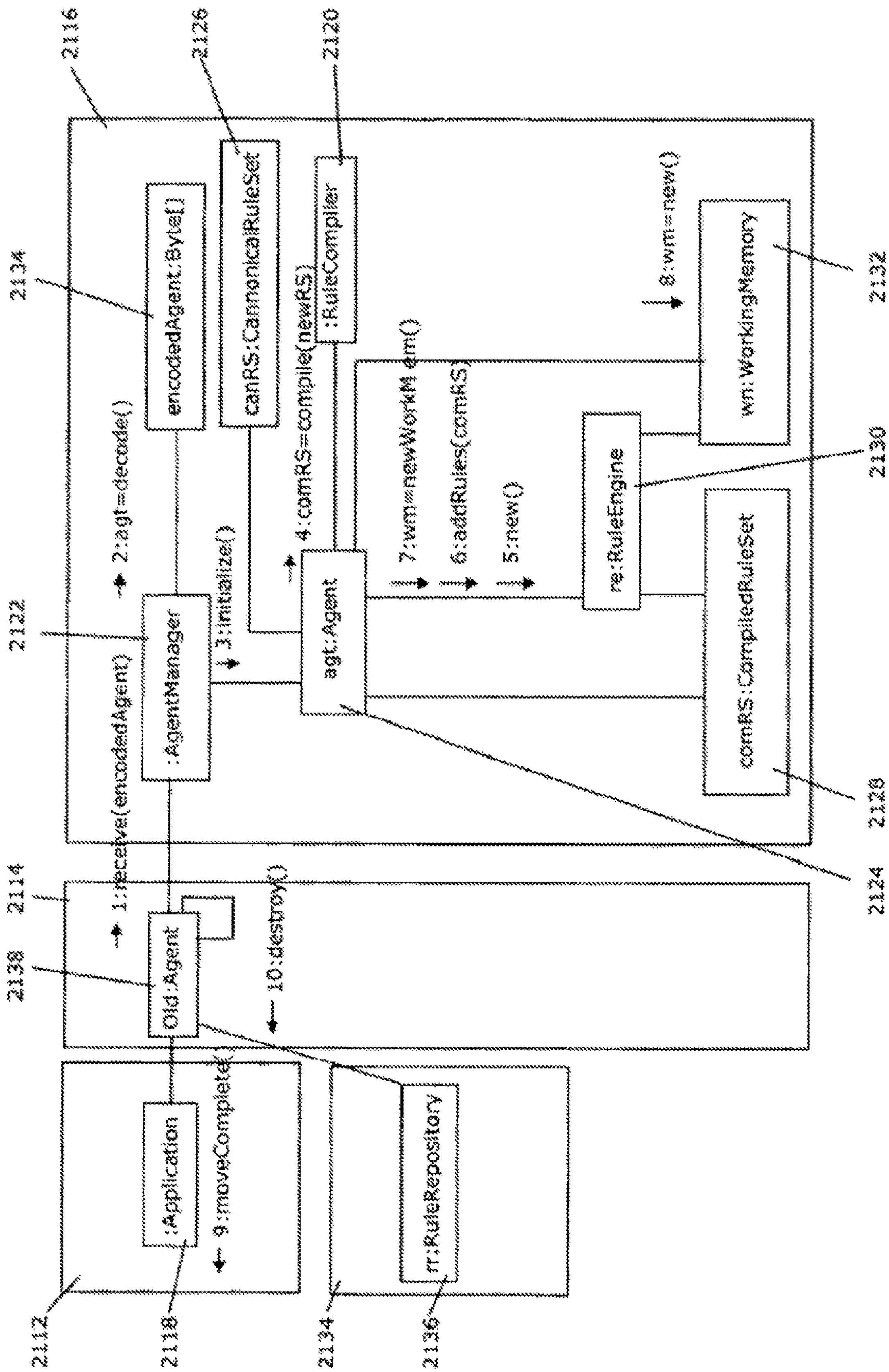
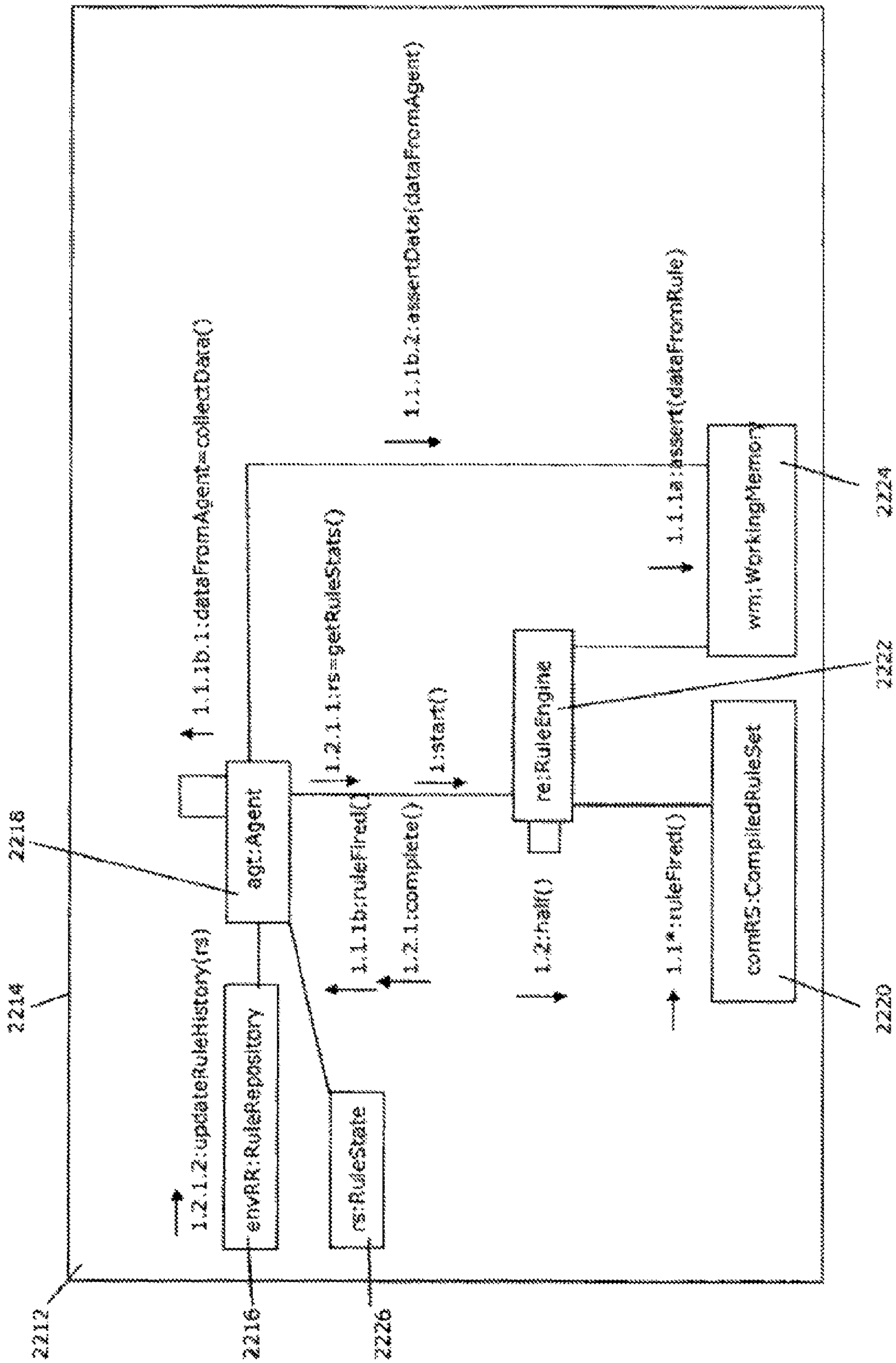

2116
2126
2120
2134
2122
2132
2130
2128
2124
2114
2138
2112
2118
2134
2136
encodedAgent:Byte[]
canRS:CannonicalRuleSet
:RuleCompiler
:AgentManager
agt:Agent
re:RuleEngine
wm:WorkingMemory
comRS:CompiledRuleSet
Old:Agent
:Application
rr:RuleRepository
1:receive(encodedAgent)
2:agt=decode()
3:initialize()
4:comRS=compile(newRS)
5:new()
6:addRules(comRS)
7:wm=newWorkMem()
8:wm=new()
9:moveComplete()
10:destroy()

Figure 21



1.1.1b.1:dataFromAgent=collectData()
1.1.1b.2:assertData(dataFromAgent)
1.2.1.1:rs=getRuleStats()
1:start()
1.1.1a:assert(dataFromRule)
agt:Agent
re:RuleEngine
wm:WorkingMemory
1.2.1.2:updateRuleHistory(rs)
1.1.1b:ruleFired()
1.2.1:complete()
1.2:halt()
1.1*:ruleFired()
envRR:RuleRepository
rs:RuleState
comRS:CompiledRuleSet
2212
2214
2216
2218
2220
2222
2224
2226

Figure 22

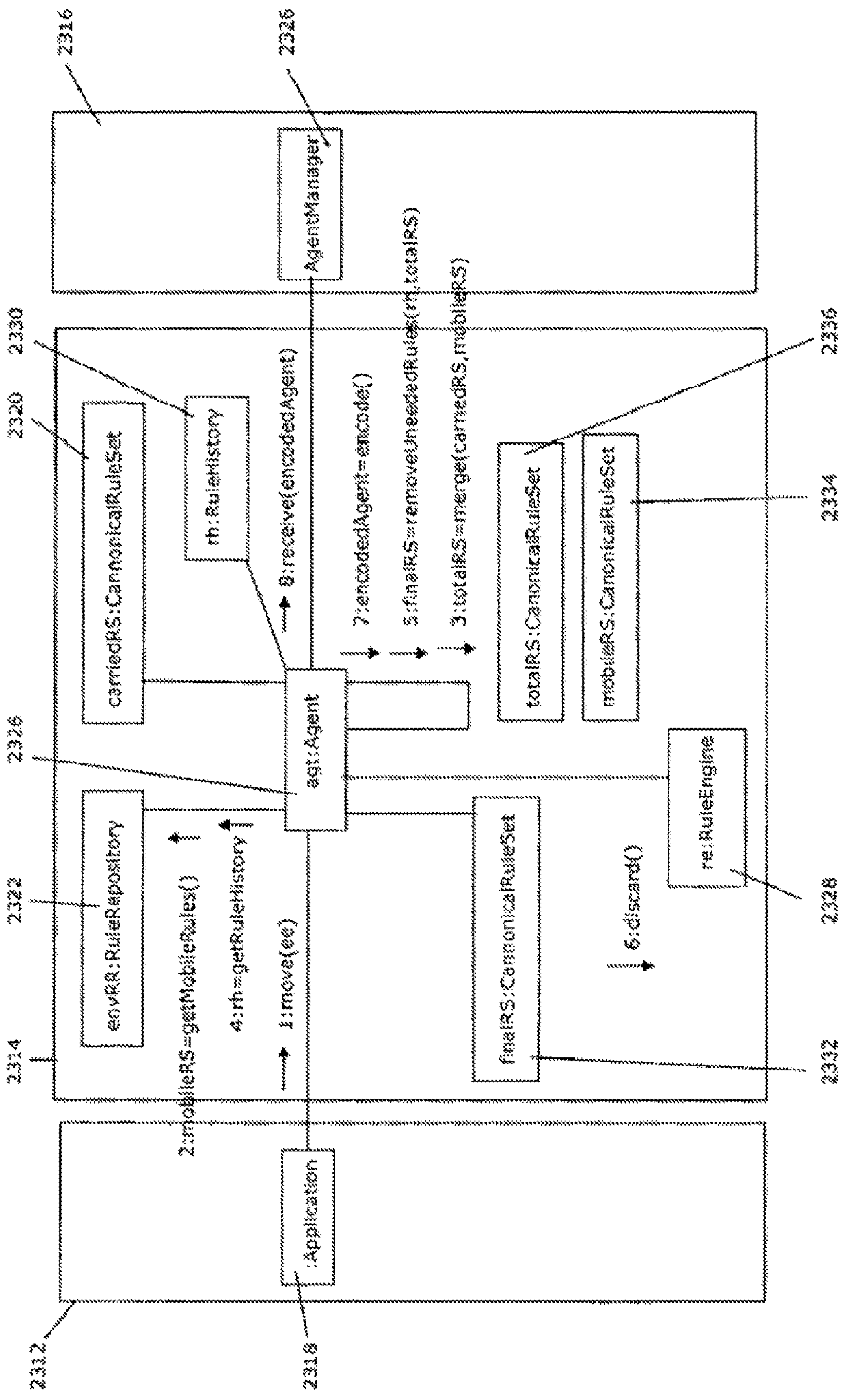

2316
2336
AgentManager
2330
2320
2338
carriedRS:CanonicalRuleSet
rh:RuleHistory
8:receive(encodedAgent)
7:encodedAgent=encode()
5:finalRS=removeUnneededRules(rh,totalRS)
3:totalRS=merge(carriedRS,mobileRS)
totalRS:CanonicalRuleSet
mobileRS:CanonicalRuleSet
2334
2326
agt:Agent
re:RuleEngine
2322
envRR:RuleRepository
2:mobileRS=getMobileRules()
4:rh=getRuleHistory
1:move(ee)
finalRS:CanonicalRuleSet
6:discard()
2328
2314
2332
:Application
2312
2318

Figure 23

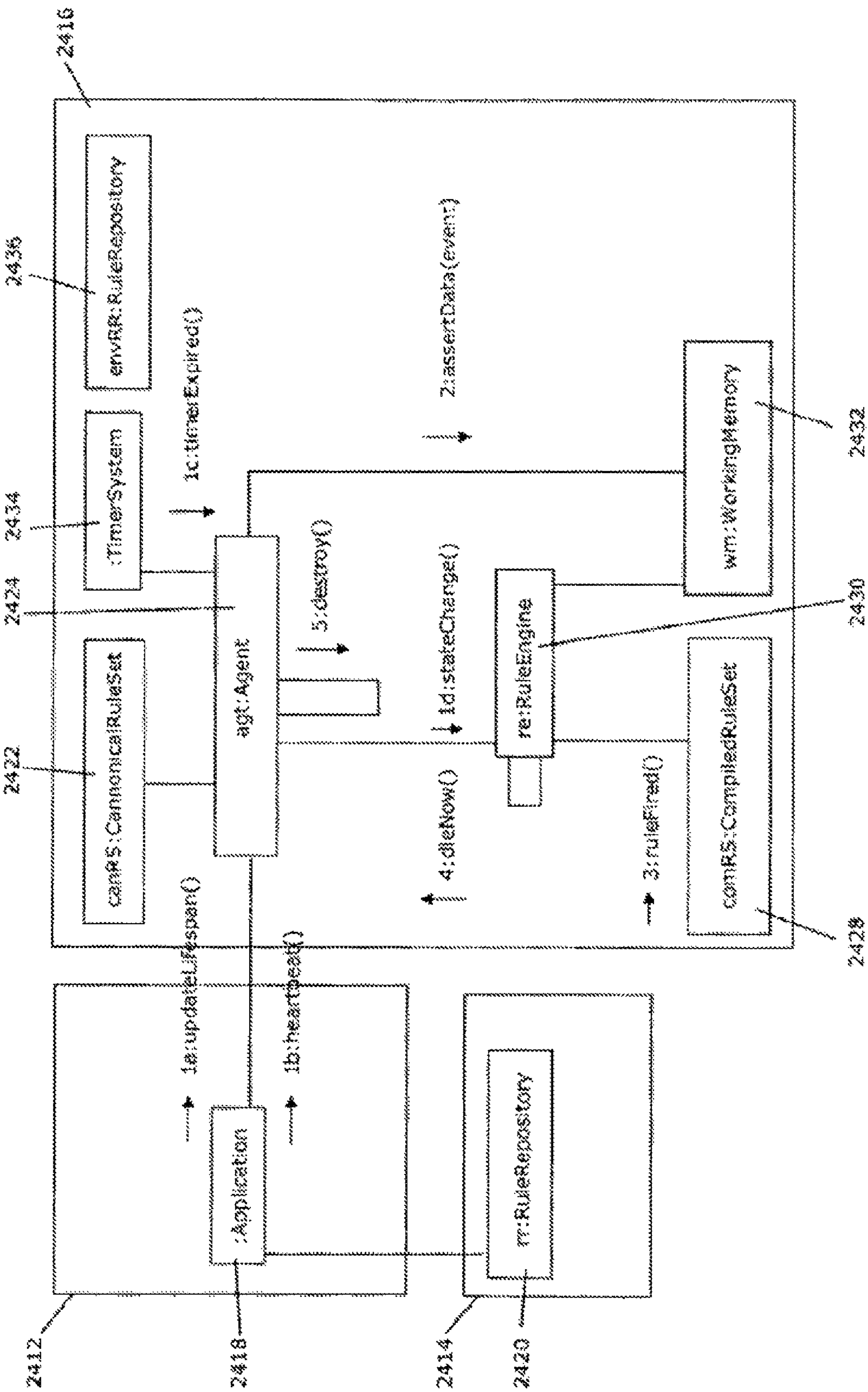

2416
envRR:RuleRepository
2436
:TimerSystem
2434
1c:timerExpired()
2:assertData(event)
wm:WorkingMemory
2432
2424
agt:Agent
5:destroy()
1d:stateChange()
re:RuleEngine
2430
canRS:CannonicalRuleSet
2422
4:dieNow()
3:ruleFired()
comRS:CompiledRuleSet
2428
1a:updateLifespan()
1b:heartbeat()
:Application
rr:RuleRepository
2412
2418
2414
2420

Figure 24

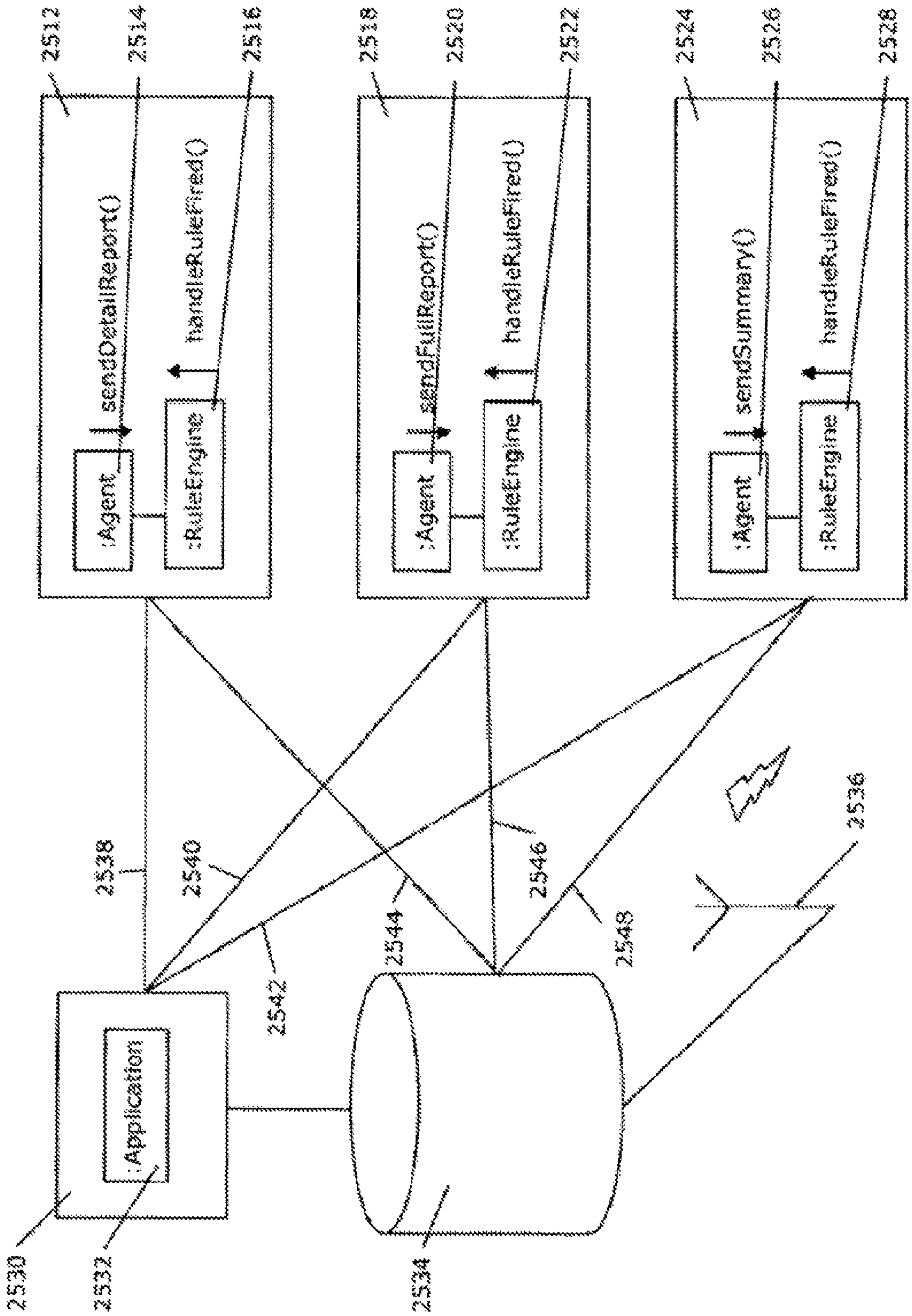

2512
2514
2516
sendDetailReport()
handleRuleFired()
:Agent
:RuleEngine
2518
2520
2522
sendFullReport()
handleRuleFired()
:Agent
:RuleEngine
2524
2526
2528
sendSummary()
handleRuleFired()
:Agent
:RuleEngine
2538
2540
2542
2544
2546
2548
2536
:Application
2530
2532
2534

Figure 25

SURVIVAL RULE USAGE BY SOFTWARE AGENTS

CROSS-REFERENCE TO RELATED PATENT APPLICATIONS

This application is a Continuation of U.S. application Ser. No. 11/645,198, filed Dec. 22, 2006, now abandoned, incorporated herein by reference in its entirety.

FIELD

The present disclosure is generally related to software agents and more particularly to software agents' use of rule-based systems.

BACKGROUND

Agents

A software agent is a software abstraction, similar to the object-oriented programming concept of an object. The concept of an agent provides a convenient and powerful way to describe a complex software entity that is capable of acting with a certain degree of autonomy in order to accomplish tasks on behalf of its user. But unlike objects, which are defined in terms of methods and attributes, an agent is defined in terms of its behavior.

Various authors have proposed different definitions of agents, commonly including concepts such as:

Persistence—code is not executed on demand but runs continuously and decides for itself when it should perform some activity

Autonomy—agents have capabilities of task selection, prioritization, goal-directed behavior, decision-making without human intervention

Social Ability—agents are able to engage other components through communication and coordination, they may collaborate on a task

Reactivity—agents perceive the context in which they operate and react to it appropriately.

Agents may also be mobile. They can move from one execution environment to another carrying both their code and their execution state. These execution environments can exist in a variety of devices in a data network including, but not limited to, servers, desktops, laptops, embedded devices, networking equipment and edge devices such as PDAs or cell phones. The characteristics of these platforms may vary widely in terms of computational capacity, networking capacity, display capabilities, etc. An agent must be able to adapt to these conditions.

Historically, agents have been programmed in a procedural manner. That is, agents are programmed with a series of steps that will ultimately result in a goal being achieved. This approach has limitations though as the logic for each agent must be compiled into the agent software and is therefore static. Complex goals can also become intractable for a programmer as the set of rules the agent must follow grows.

Rule-Based Systems

In his tutorial, Introduction to Rule-Based Systems, James Freeman-Hargis defines a rule-based system to consist of a set of assertions and a set of rules for how to act on the assertion set. When a set of data is supplied to the system, it may result in zero or more rules firing. Rule based systems are rather simplistic in nature, consisting of little more than a group of if-then statements, but form the basis of many "expert systems." In an expert system, the knowledge of an expert is encoded into the rule-set. When a set of data is supplied to the system, the system will come to the same conclusion as the expert. With this approach there is a clear separation between the domain logic (a rule set) and the execution of the agent. As mentioned, the procedural agent approach tightly couples the two.

The rule-based system itself uses a simple technique. It starts with a rule-set, which contains all of the appropriate knowledge encoded into If-Then rules, and a working memory, which mayor may not initially contain any data, assertions or initially known information. The system in operation examines all the rule conditions (IF) and determines a subset, the conflict set, of the rules whose conditions are satisfied based on the working memory. Of this conflict set, one of those rules is triggered (fired). The rule that is chosen is based on a conflict resolution strategy. When the rule is fired, any actions specified in its THEN clause are carried out. These actions can modify the working memory, the rule-set itself, or do just about anything else the system programmer decides to include. This loop of firing rules and performing actions continues until one of two conditions are met: there are no more rules whose conditions are satisfied or a rule is fired whose action specifies the rule engine execution should terminate.

Rule-based systems, as defined above, are adaptable to a variety of problems. In some problems, working memory asserted data is provided with the rules and the system follows them to see where they lead. This approach is known as forward-chaining. An example of this is a medical diagnosis in which the problem is to diagnose the underlying disease based on a set of symptoms (the working memory). A problem of this nature is solved using a forward-chaining, data-driven, system that compares data in the working memory against the conditions (IF parts) of the rules and determines which rules to fire.

In other problems, a goal is specified and the system must find a way to achieve that specified goal. This is known as backward-chaining. For example, if there is an epidemic of a certain disease, this system could presume a given individual had the disease and attempt to determine if its diagnosis is correct based on available information. A backward-chaining, goal-driven, system accomplishes this. To do this, the system looks for the action in the THEN clause of the rules that matches the specified goal. In other words, it looks for the rules that can produce this goal. If a rule is found and fired, it takes each of that rule's conditions as goals and continues until either the available data satisfies all of the goals or there are no more rules that match.

The Rete algorithm is an efficient pattern matching algorithm for implementing forward-chaining, rule-based systems. The Rete algorithm was designed by Dr. Charles L. Forgy of Carnegie Mellon University in 1979. Rete has become the basis for many popular expert systems, including JRules, OPS5, CLIPS, JESS, Drools, and LISA.

A naïve implementation of a rule-based system might check each rule against the known facts in the knowledge base, firing that rule if necessary, then moving on to the next rule (and looping back to the first rule when finished). For even moderate sized rules and fact knowledge-bases, this naïve approach performs far too slowly.

The Rete algorithm (usually pronounced either 'REET' or 'REE-tee', from the Latin 'rete' for net, or network) provides the basis for a more efficient implementation of an expert system. A Rete-based expert system builds a network of nodes, where each node (except the root) corresponds to a pattern occurring in the left-hand-side of a rule. The path from the root node to a leaf node defines a complete rule left-hand-side. Each node has a memory of facts which satisfy that pattern.

As new facts are asserted or modified, they propagate along the network, causing nodes to be annotated when that fact matches that pattern. When a fact or combination of facts causes all of the patterns for a given rule to be satisfied, a leaf node is reached and the corresponding rule is triggered.

The Rete algorithm is designed to sacrifice memory for increased speed. In most cases, the speed increase over naive implementations is several orders of magnitude (because Rete performance is theoretically independent of the number of rules in the system). In very large systems, however, the original Rete algorithm tends to run into memory consumption problems which have driven the design of Rete variants.

Therefore, what is needed is an ability to utilize a set of survival rules by an agent. More specifically what is needed is utilization of survival rules by an agent that uses a rule engine in an execution environment.

SUMMARY OF THE INVENTION

The present invention provides a system, method, and computer readable medium for utilization of survival rules by an agent that uses a rule engine in an execution environment.

In one embodiment of the present invention, a method for determining the lifespan of an agent utilizing a rule engine and a set of canonical survival rules, in an execution environment comprising collecting a survival rule, asserting a survival data into a working memory and executing the rule engine with the set of survival rules and the working memory. The method may also comprise supplying the agent with the survival rule during construction. The method may additionally comprise supplying the agent with the survival rule during movement from a first execution environment to a second execution environment. The method may further comprise collecting the survival rule when the agent that is being moved from a first execution environment to a second execution environment arrives in the second execution environment. The method may also comprise collecting the survival rule from a rule repository in the execution environment. The method may also comprise the survival rule being specific to the agent. The method may additionally comprise the survival rule being specific to the execution environment. The method may further comprise asserting the survival data in the working memory based on a state change in the agent. The method may also comprise asserting the survival data in the working memory based on a timer event in the execution environment. The method may further comprise asserting the survival data in the working memory based on firing the survival rule by the rule engine.

In another embodiment of the present invention, a computer readable medium for determining the lifespan of an agent utilizing a rule engine and a set of canonical survival rules in a first execution environment comprising instructions for asserting an initial set of survival data into a working memory, collecting the set of survival rules, compiling the set of survival rules, executing the rule engine with the set of survival rules and a working memory, asserting the survival data into the working memory and firing a survival rule by the rule engine. The computer readable medium may also comprise instructions for collecting the set of survival rules from a rule repository in a second execution environment. The computer readable medium may additionally comprise instructions for asserting a survival data into the working memory as a result of firing the survival rule that requires extending the life of the agent. The computer readable medium may further comprise instructions for updating a lifespan timer in the execution environment of the agent as a result of firing the survival rule that requires extending the life of the agent. The computer readable medium may also comprise instructions for destroying the agent as a result of firing the survival rule that requires termination of the agent. The computer readable medium may additionally comprise instructions for capturing rule statistics as a result of firing the survival rule that requires termination of the agent. The computer readable medium may further comprise instructions for asserting the survival data in the working memory based on receiving a lifespan update request from an application. The computer readable medium may also comprise instructions for updating a lifespan timer in the execution environment of the agent based on receiving a lifespan update request from an application.

In a further embodiment of the present invention, a system for determining the lifespan of an agent utilizing a rule engine and a set of canonical survival rules in a first execution environment comprising a working memory for the agent in the first execution environment and a first processor communicably coupled to the first memory, wherein the processor collects the set of canonical survival rules, compiles the set of survival rules, executes the rule engine with the set of compiled survival rules and the working memory, fires a survival rule, asserts a survival data into the working memory as a result of the rule firing, and sends a heartbeat to an originating application as a result of the rule firing. The system may further comprise a second memory stores the originating application and a second processor communicably coupled to the second memory, wherein the second processor executes the originating application that receives the heartbeat.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a diagram illustrating an exemplary process of constructing an agent locally with a set of canonical rules supplied during construction;

FIG. **2** is a diagram illustrating an exemplary process of constructing an agent remotely with a set of canonical rules supplied during construction;

FIG. **3** is a diagram illustrating an exemplary process of constructing an agent in a remote execution environment during which a set of canonical rules is retrieved from outside the execution environment;

FIG. **4** is a diagram illustrating an exemplary process of moving an agent carrying canonical rules from a first execution environment;

FIG. **5** is a diagram illustrating an exemplary process of moving an agent carrying canonical rules to a second execution environment;

FIG. **6** is a diagram illustrating an exemplary process of an agent utilizing a rule-based system engine for execution;

FIG. **7** is a diagram illustrating an exemplary process of constructing an agent locally with a set of compiled rules supplied during construction;

FIG. **8** is a diagram illustrating an exemplary process of constructing an agent remotely with a set of compiled rules supplied during construction;

FIG. **9** is a diagram illustrating an exemplary process of constructing an agent remotely during which a set of compiled rules that are retrieved from outside the execution environment;

FIG. **10** is a diagram illustrating an exemplary process of moving an agent carrying compiled rules from a first execution environment;

FIG. **11** is a diagram illustrating an exemplary process of moving an agent carrying compiled rules to a second execution environment;

FIG. **12** is a diagram illustrating an exemplary process of constructing an agent remotely with a set of canonical rules carried by the agent and a set of canonical execution environment rules resident in a remote environment;

FIG. **13** is a diagram illustrating an exemplary process of constructing an agent remotely with a set of canonical rules fetched by the agent and a set of canonical execution environment rules resident in a remote environment;

FIG. **14** is a diagram illustrating an exemplary process of moving an agent carrying canonical rules from a first execution environment that includes execution environment rules;

FIG. **15** is a diagram illustrating an exemplary process of moving an agent carrying canonical rules to a second execution environment that includes a repository of canonical execution environment rules;

FIG. **16** is a diagram illustrating an exemplary process of constructing an agent at a remote location with an as-needed set of canonical rules supplied during construction;

FIG. **17** is a diagram illustrating an exemplary process of constructing an agent at a remote location with an as-needed set of canonical rules fetched during construction;

FIG. **18** is a diagram illustrating an exemplary process of moving an agent with supplied as-needed canonical rules from a first execution environment;

FIG. **19** is a diagram illustrating an exemplary process of moving an agent with supplied as-needed canonical rules to a second execution environment;

FIG. **20** is a diagram illustrating an exemplary process of moving an agent from a first execution environment with a fetched as-needed set of canonical rules;

FIG. **21** is a diagram illustrating an exemplary process of moving an agent to a second execution environment with a fetched as-needed set of canonical rules;

FIG. **22** is a diagram illustrating an exemplary process of a rule-based agent updating rule history when rule processing is halted in an execution environment;

FIG. **23** is a diagram illustrating an exemplary process of a rule-based agent identifying and carrying only needed canonical rules during as part of movement to another execution environment;

FIG. **24** is a diagram illustrating an exemplary process of an agent using a set of survival rules to determine its lifespan; and

FIG. **25** is a diagram illustrating an exemplary process of an agent using a set of data narrowing rules to determine how much data should be sent over the network.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Construction

Agents which utilize rule based systems may be constructed locally or remotely. In order to operate, these agents need an initial set of canonical rules that can be compiled and loaded into an associated rule engine. These rules can either be supplied at construction or a rule repository location can be supplied so that the rules may be fetched during construction or at a later time.

Referring now to FIG. **1**, a diagram illustrating an exemplary process of constructing an agent locally with a set of canonical rules supplied during construction is shown. An application **110**, in an execution environment **112**, requests a set of rules for an agent from a rule repository **116** based on the goals of the agent that is being created. The result is a collection of canonical rules, known as a rule set **118**. The rule set **118** is passed to the agent **120** during construction. The agent **120** takes the rule set **118** and requests that it be compiled by the local rule compiler **122**. This results in the creation of a compiled rule set **124**. At this point the agent creates the rule engine **126** that will be used to execute the rule set. Note that if the execution environment **112** includes a rule engine, then one may not need to be created. After the rule engine **126** is created or located, the agent **120** supplies the engine **126** with the compiled rule set **124**. Finally, the agent **120** requests a new working memory **128** from the rule engine **126**. The working memory will hold all of the data the agent chooses to assert before and during execution of the rule engine. At this point, the agent **120** is ready to be moved to another execution environment or to execute the rule engine. Both of these processes are described in detail in later sections.

Referring now to FIG. **2**, a diagram illustrating an exemplary process of constructing an agent remotely with a set of canonical rules supplied during construction is shown. An application **218**, in execution environment **212**, requests a set of rules for an agent from a rule repository **220** in execution environment **214** based on the goals of the agent that is being created. The result is a collection of canonical rules, known as a rule set **222**. The rule set **222** is passed to the agent **224** during construction in execution environment **216**. The agent **224** in execution environment **216** takes the rule set **222** and requests that it be compiled by the local rule compiler **226**. This results in the creation of a compiled rule set **228**. At this point the agent creates the rule engine **230** that will be used to execute the rule set. Note that if execution environment **216** includes a rule engine, then one may not need to be created. After the rule engine **230** is created or located, the agent **224** supplies the engine **230** with the compiled rule set **228**. Finally, the agent **224** requests a new working memory **232** from the rule engine **230**. The working memory will hold all of the data the agent chooses to assert before and during execution of the rule engine. At this point, the agent **224** is ready to be moved to another execution environment or to execute the rule engine.

Referring now to FIG. **3**, a diagram illustrating an exemplary process of constructing an agent in a remote execution environment during which a set of canonical rules is retrieved from outside the execution environment is shown. An application **318**, in execution environment **312**, requests the creation of an agent **324** in execution environment **316**. Agent **324** is passed the location of a rule repository **320** during construction. During construction, the agent **324** requests a set of rules based on its goals from the rule repository **320** in execution environment **314**. The result is a collection of canonical rules, known as a rule set **322**. The agent **324** in execution environment **316** takes the rule set **322** and requests that it be compiled by the local rule compiler **326**. This results in the creation of a compiled rule set **328**. At this point the agent creates the rule engine **330** that will be used to execute the rule set. Note that if execution environment **314** includes a rule engine, then one may not need to be created. After the rule engine **330** is created or located, the agent **324** supplies the engine **330** with the compiled rule set **328**. Finally, the agent **324** requests a new working memory **332** from the rule engine **330**. The working memory will hold all of the data the agent chooses to assert before and during execution of the rule engine. At this point, the agent **324** is ready to be moved to another execution environment or to execute the rule engine.

Movement

An agent may move from one execution environment to another. This process may be initiated by a variety of means including but not limited to an application, another agent, another object, the existing agent itself, a human interacting with the execution environment or a rule executing in the agent's rule engine.

Referring now to FIGS. **4** and **5**, diagrams illustrating an exemplary process of moving an agent carrying canonical rules from one execution environment to another are shown. An application **418** in execution environment **412** requests that an agent **424** in execution environment **414** move to execution environment **416**. The location of execution environment **416** may be described in the move request by an IP address and port, Uniform Resource Locator (URL), or any other means of addressing. The agent **424** discards its rule engine **430** along with the associated compiled rule set **428** and working memory **432**. The agent **424** then encodes itself along with its canonical rule set **422** into a transferable form **434**. Though a byte array is shown, the encoded agent could take any form that can be transferred between the two execution environments. Once the agent **424** has created an encoded version of itself **434** in execution environment **414** it transfers the encoded version **434** to an agent manager **426** residing in execution environment **416**.

Referring now to FIG. **5**, the process continues with the agent manager **522** receiving the encoded agent **534**. Upon receipt of the encoded agent **534**, the agent manager **522** decodes the encoded agent **534** into a new version of the agent **524** and the agent's canonical rule set **526** in execution environment **516**. Once the agent **524** and rule set **526** have been materialized, the agent manager **522** requests that the agent **524** initialize. This request prompts the agent **524** to go to the execution environment's rule compiler **520** and request compilation of its canonical rule set **526**. The result is a compiled rule set **528**. The agent then creates a new rule engine **530** and subsequently passes the compiled rule set **528** to it. As during construction, if the execution environment has a rule engine, then one may not need to be created. Once the engine **530** has been located/created and the compiled rule set **528** has been added to it, the agent **524** requests a new working memory from the rule engine. As before, the working memory will hold all of the data the agent chooses to assert before and during execution of the rule engine. At this point, the agent **524** is ready to execute the rule engine. Once the move operation completes, the old version of the agent **518** in execution environment **514** indicates to the requesting application **518** in execution environment **512** that the move operation has completed. Once the notification has been made, the old agent **534** is destroyed.

Execution

Once an agent has been initialized in an execution environment through either creation or movement, it can be sent requests to perform different tasks. These tasks mayor may not require sending one or more responses. Recall that during construction an agent is associated with a newly created or resident rule engine and that a rule set is provided to that engine.

Referring now to FIG. **6**, a diagram illustrating an exemplary process of an agent utilizing a rule-based system engine for execution is shown. An application **616** in execution environment **612** sends a request to an agent **618** in execution environment **614**. Upon receiving the request, the agent **618**, collects an initial set of data and asserts it into its working memory **624** in order to accomplish the task requested. Note that this data may be collected from the local execution environment, from an accessible database, from other objects, from other agents, from a human via a man machine interface, from a computer readable medium or any combinations of the above. With a provided compiled rule set **620**, and an initial set of data in working memory **624**, the rule engine **622** is then started by the agent **618**.

When the engine **622** starts, it processes the objects in working memory against the rule set **620**. This may result in one or more rules being fired by the engine **622**. When a rule is fired it may add, modify or delete objects in working memory **624**. Additionally, the engine **622** can inform the agent **618** which may result in a number of actions being taken by the agent **618** including, but not limited to, the collection and assertion of additional data into the working memory **624** (shown) and/or sending of a preliminary response back to the application. This sequence will continue until the task is completed, there are no more rules available to fire, or the agent receives an event, such as move or terminate, causing it to halt rule engine processing. Upon completion of the task, the agent **618** may send a response back to the application **616** that initiated the request (shown).

Pre-Compiled Agent Rule Set Usage

As noted above, the process of adding rules to the rule engine can be expensive in terms of CPU utilization on the execution environment in which the operation is performed. This can be problematic for less powerful hosts such as personal devices (cell phones, PDAs, etc.) and servers with limited available CPU resources. Therefore, another embodiment of the invention creates the compiled rule set in the execution environment of the application that creates an agent instead of in the environment in which the agent is constructed or moved.

Referring now to FIG. **7**, a diagram illustrating an exemplary process of constructing an agent locally with a set of compiled rules supplied during construction is shown. An application **712**, in execution environment **714**, requests a set of rules for an agent from a rule repository **720** based on the goals of the agent that is being created. The result is a collection of canonical rules, known as a rule set **724**. The application **712** takes the rule set **724** and requests that it be compiled by the local rule compiler **722**. This results in the creation of a compiled rule set **724**. The rule set **724** is passed to the agent **718** during construction. At this point the agent creates the rule engine **726** that will be used to execute the rule set. Note that if the execution environment **714** includes a rule engine, then one may not need to be created. After the rule engine **726** is created or located, the agent **722** supplies the engine **726** with the compiled rule set **724**. Finally, the agent **110** requests a new working memory **728** from the rule engine **726**. The working memory will hold all of the data the agent chooses to assert before and during execution of the rule engine. At this point, the agent **718** is ready to be moved to another execution environment or to execute the rule engine.

Referring now to FIG. **8**, a diagram illustrating an exemplary process of constructing an agent remotely with a set of compiled rules supplied during construction is shown. An application **812**, in execution environment **814**, requests a set of rules for an agent from a rule server **828** in execution environment **818** based on the goals of the agent that is being created. The rule server **828** queries a rule repository **830** for the rules. The result is a collection of canonical rules, known as a rule set **832**. The rule server **828** in execution environment **202** takes the rule set **832** and requests that it be compiled by the local rule compiler **834**. This results in the creation of a compiled rule set **826**. The compiled rule set **826** is passed to the agent **820** during construction in execution environment **204**. At this point, the agent **820** creates the rule engine **822** that will be used to execute the rule set. Note that if execution environment **816** includes a rule engine, then one may not need to be created. After the rule engine **822** is created or located, the agent **820** supplies the engine **822** with the compiled rule set **826**. Finally, the agent **820** requests a new working memory **116** from the rule engine **822**. The working memory will hold all of the data the agent chooses to assert before and during execution of the rule engine. At this point, the agent **820** is ready to execute the rule engine.

Referring now to FIG. **9**, a diagram illustrating an exemplary process of constructing an agent in a remote execution environment during which a set of compiled rules is retrieved from outside the execution environment is shown. An application **912**, in execution environment **914**, requests the creation of an agent **920** in execution environment **916**. Agent **920** is passed the location of a rule server **928**, resident in execution environment **918**, during construction. During construction, the agent **920** requests a set of compiled rules based on its goals from the rule server **928** in execution environment **918**. The rule server **928** queries a rule repository **930** for a set of rules. The result is a collection of canonical rules, known as a rule set **932**. The rule server **928** in execution environment **918** takes the rule set **932** and requests that it be compiled by the local rule compiler **934**. This results in the creation of a compiled rule set **926**. At this point the agent **920** creates a rule engine **922** that will be used to execute the rule set. Note that if execution environment **916** includes a rule engine, then one may not need to be created. After the rule engine **922** is created or located, the agent **920** supplies the engine **922** with the compiled rule set **926**. Finally, the agent **920** requests a new working memory **924** from the rule engine **922**. The working memory will hold all of the data the agent chooses to assert before and during execution of the rule engine. At this point, the agent **920** is ready to execute the rule engine.

Referring now to FIGS. **10-11**, diagrams illustrating an exemplary process of moving an agent carrying compiled rules from one execution environment to another are shown. An application **1018** in execution environment **1012** request that an agent **1022** in execution environment **1014** move to execution environment **1016**. The location of execution environment **1016** may be described in the move request by an IP address and port, Uniform Resource Locator (URL), or any other means of addressing. The agent **1022** discards its rule engine **1030** along with the associated working memory **1032**. Subsequently, the agent **1022** discards its canonical rule set **1020** if it still has a reference to it. The agent **1022** then encodes itself along with its compiled rule set **1028** into a transferable form **1024**. Though a byte array is shown, the encoded agent could take any form that can be transferred between the two execution environments. Once the agent **1022** has created an encoded version of itself **1024** in execution environment **1014** it transfers the encoded version **1024** to an agent manager **1026** residing in execution environment **1016**.

Referring now to FIG. **11**, the process continues with an agent manager **1122** receiving an encoded agent **1134**. Upon receipt of the encoded agent **1134**, the agent manager **1122** decodes the encoded agent **1134** into a new version of the agent **1124** and its compiled rule set **1128** in execution environment **1116**. Once the agent **1124** and rule set **1128** have been decoded, the agent manager **1122** requests that the agent **1124** initialize. This request prompts the agent **1124** to create a new rule engine **1130** and subsequently pass the compiled rule set **1128** to it. As during construction, if the execution environment has a rule engine, then one may not need to be created. Once the engine **1130** has been located/created and the compiled rule set **1128** has been added to it, the agent **1124** requests a new working memory **1132** from the rule engine. As before, the working memory will hold all of the data the agent chooses to assert before and during execution of the rule engine. At this point, the agent **1124** is ready to execute the rule engine. Once the move operation completes, the old version of the agent **1118** in execution environment **1114** indicates to the requesting application **1118** in execution environment **1112** that the move operation has completed. Once the notification has been made, the old agent **1118** is destroyed.

Execution Environment Rule Set Usage

Each execution environment may have access to a local rule repository which allow for the rules for a particular domain, domain rules, to be distributed, or partitioned, in any number of rule repositories. An agent may be configured to only use rules provided at construction essentially ignoring rules available from each execution environment's local rule repository. The more general case is for the agent to make use of the rules that it carries with itself along with the rules extracted from the execution environment's local rule repository. Local rule repositories may contain rules for several different domains and are usually specific to execution environment objects that will be asserted to working memory but may also apply to execution environment concerns such as security, resource usage, scheduling, or any other execution environment attribute.

Referring now to FIG. **12**, a diagram illustrating an exemplary process of constructing an agent remotely with a set of canonical rules carried by the agent and a set of canonical rules resident in a remote environment is shown. An application **1218**, in execution environment **1212**, requests a set of rules for an agent from a rule repository **1220** in execution environment **1214** based on the goals of the agent that is being created. The result is a collection of canonical rules, known as a rule set **1230**. The rule set **1230** is passed to the agent **1232** during construction in execution environment **1216**. During construction, the agent **1232** requests the set of rules from a local rule repository **1234** given the agent's domain (not shown). The result of which, canonical rule set **1236**, is then merged with the construction supplied rule set **1230** to form a merged rule set **1222**. This rule set contains all the domain and environment specific rules that the agents' rule engine will execute. The agent **1232** then takes the merged rule set **1222** and requests that it be compiled by the local rule compiler **1226**. This results in the creation of a compiled rule set **1238**. At this point the agent creates a rule engine **1224** that will be used to execute the rule set **1238**. Note that if execution environment **1216** includes a rule engine, then one may not need to be created. After the rule engine **1224** is created or located, the agent **1232** supplies the engine **1224** with the compiled rule set **1238**. Finally, the agent **1232** requests a new working memory **1228** from the rule engine **1224**. The working memory will hold all of the data the agent chooses to assert before and during execution of the rule engine.

Referring now to FIG. **13**, a diagram illustrating an exemplary process of constructing an agent remotely with a set of canonical rules fetched by the agent and a set of canonical local rules resident in a remote environment is shown. An application **1318**, in execution environment **1312**, requests the creation of an agent **1332** in execution environment **1316**. Agent **1332** is passed the location of a rule repository **1320** during construction. During construction, the agent **1332** requests a set of rules based on its goals from the rule repository **1320** in execution environment **1314**. The result is a collection of canonical rules, known as a rule set **1330**. During construction, the agent **1332** requests the set of rules from a local rule repository **1334** that apply to its domain. The result of which, canonical rule set **1336**, is then merged with the fetched rule set **104** to form a merged rule set **1322**. This rule set contains all the domain and environment specific rules that the agents' rule engine will execute. The agent **1332** then takes the merged rule set **1322** and requests that it be compiled by the local rule compiler **1326**. This results in the creation of a compiled rule set **1338**. At this point the agent creates a rule engine **1324** that will be used to execute the rule set **1338**. Note that if execution environment **1316** includes a rule engine, then one may not need to be created. After the rule engine **1324** is created or located, the agent **1332** supplies the engine **1324** with the compiled rule set **1338**. Finally, the agent **1332** requests a new working memory **1328** from the rule engine **1324**. The working memory will hold all of the data the agent chooses to assert before and during execution of the rule engine.

Referring now to FIGS. **14-15**, diagrams illustrating an exemplary process of moving an agent carrying canonical rules to an execution environment that includes a local repository of canonical rules are shown. Referring now to FIG. **14**, an application **1418** in execution environment **1412** requests that an agent **1422** in execution environment **1414** move to execution environment **1416**. The location of execution environment **1416** may be described in the move request by an IP address and port, Uniform Resource Locator (URL), or any other means of addressing. The agent **1422** discards its rule engine **1430** along with the associated compiled rule set **1428** and working memory **1432**. The agent **1422** then encodes itself along with its canonical rule set **1420** into a transferable form **1424**. Though a byte array is shown, the encoded agent could take any form that can be transferred between the two execution environments. Once the agent **1422** has created an encoded version of itself **1424** in execution environment **1414** it transfers the encoded version **1424** to an agent manager **1426** residing in execution environment **1416**.

Referring now to FIG. **15**, the process continues with the agent manager **1522** receiving the encoded agent **1534**. Upon receipt of the encoded agent **1534**, the agent manager **1522** decodes the encoded agent **1534** into a new agent **1526** and its canonical rule set **1540** in execution environment **1516**. Once the agent **1526** and rule set **1540** have been decoded, the agent manager **1522** requests that the agent **1526** initialize. This request prompts the agent **1526** to request the set of rules applicable to the agent's domain from a local rule repository **1536**. The result of which, canonical rule set **1538**, is then merged with the carried rule set **1540** to form a merged rule set **1534**. This rule set contains all the domain and environment specific rules that the agents rule engine will execute. The agent **1526** then takes the merged rule set **1534** and requests that it be compiled by the local rule compiler **1524**. The result is a compiled rule set **1528**. The agent then creates a new rule engine **1530** and subsequently passes the compiled rule set **1528** to it. As during construction, if the execution environment has a sharable rule engine, then one may not need to be created. Once the engine **1530** has been located/created and the compiled rule set **1528** has been added to it, the agent **1526** requests a new working memory **1532** from the rule engine. As before, the working memory will hold all of the data the agent chooses to assert before and during execution of the rule engine. Once the move operation completes, the old version of the agent **1520** in execution environment **1514** indicates to the requesting application **1518** in execution environment **1512** that the move operation has completed. Once the notification has been made, the old agent **1520** is destroyed.

As-Needed Rules

As there is a cost of carrying around unnecessary rules in terms of both CPU and memory usage, it is desirable in many cases to supply an agent with a subset of its total potential rule set. This can be done in a context-specific manner based on the goals and execution environment of the agent. For example, if each device upon which an agent will be executing only contains a small screen, then there is no need to carry the rules for display on a standard computer monitor. As another example, an agent who moves progressively further in a single direction, perhaps among GPS enabled fixed location devices, need not carry rules that only apply to previous GPS locations.

Referring now to FIG. **16**, a diagram illustrating an exemplary process of constructing an agent at a remote location with an as-needed set of canonical rules supplied during construction is shown. An application **1618**, in execution environment **1612**, requests a set of rules for an agent from a rule repository **1620** in execution environment **1614** based on the goals and initial execution environment of the agent that is being created. When supplied with a target execution environment, the rule repository **1620** can filter out rules that do not apply to that type of environment. The result is a collection of canonical rules, known as a rule set **1622**. The rule set **1622** is passed to the agent **1624** during construction in execution environment **1616**. The agent **1624** in execution environment **1616** takes the rule set **1622** and requests that it be compiled by the local rule compiler **1626**. This results in the creation of a compiled rule set **1628**. At this point the agent creates the rule engine **1630** that will be used to execute the rule set. Note that if execution environment **1616** includes a rule engine, then one may not need to be created. After the rule engine **1630** is created or located, the agent **1624** supplies the engine **1630** with the compiled rule set **1628**. Finally, the agent **1624** requests a new working memory **1632** from the rule engine **1630**. The working memory will hold all of the data the agent chooses to assert before and during execution of the rule engine. At this point, the agent **1624** is ready to be moved to another execution environment or to execute the rule engine.

Referring now to FIG. **17**, a diagram illustrating an exemplary process of constructing an agent at a remote location with an as-needed set of canonical rules fetched during construction is shown. An application **1718**, in execution environment **1712**, requests the creation of an agent **1724** in execution environment **1716**. Agent **1724** is passed the location of a rule repository **1720** during construction. During construction, the agent **1724** requests a set of rules based on its goals and execution environment from the rule repository **1720** in execution environment **1714**. When supplied with the target execution environment, the rule repository **1720** can filter out rules that do not apply to that type of environment. The result is a collection of canonical rules, known as a rule set **1722**. The agent **1724** in execution environment **204** takes the rule set **1722** and requests that it be compiled by the local rule compiler **1726**. This results in the creation of a compiled rule set **1728**. At this point the agent creates the rule engine **1730** that will be used to execute the rule set. Note that if execution environment **1714** includes a rule engine, then one may not need to be created. After the rule engine **1730** is created or located, the agent **1724** supplies the engine **1730** with the compiled rule set **1728**. Finally, the agent **1724** requests a new working memory **1732** from the rule engine **1730**. The working memory will hold all of the data the agent chooses to assert before and during execution of the rule engine. At this point, the agent **1724** is ready to be moved to another execution environment or to execute the rule engine.

Referring now to FIGS. **18-19**, diagrams illustrating an exemplary process of moving an agent from one execution environment to another with a supplied as-needed set of canonical rules are shown. An application **1818** in execution environment **1812** requests that an agent **1822** in execution environment **1814** move to execution environment **1816**. The location of execution environment **1816** may be described in the move request by an IP address and port, Uniform Resource Locator (URL), or any other means of addressing. The move request includes a new as-needed canonical rule set **1834** based on the agent's goals and target execution environment. The agent **1822** discards its rule engine **1830** along with the associated compiled rule set **1828** and working memory **1832**. In addition the agent **1822** discards its old canonical rule set **1820**. At this point, the agent **1822** encodes itself along with its new as-needed canonical rule set **1834** into a transferable form **1824**. Though a byte array is shown, the encoded agent could take any form that can be transferred between the two execution environments. Once the agent **1822** has created an encoded version of itself **1824** in execution environment **1814** it transfers the encoded version **1824** to an agent manager **1826** residing in execution environment **1816**.

Referring now to FIG. **19**, the process continues with the agent manager **1922** receiving an encoded agent **1934**. Upon receipt of the encoded agent **1934**, the agent manager **118** decodes the encoded agent **1934** into a new version of the agent **1924** and its new canonical rule set **1926** in execution environment **1916**. Once the agent **1924** and rule set **1926** have been materialized, the agent manager **1922** requests that the agent **1922** initialize. This request prompts the agent **1922** to go to the execution environments' rule compiler **1920** and request compilation of its canonical rule set **1926**. The result is a compiled rule set **1928**. The agent then creates a new rule engine **1930** and subsequently passes the compiled rule set **1928** to it. As during construction, if the execution environment has a rule engine, then one may not need to be created. Once the engine **1928** has been located/created and the compiled rule set **1926** has been added to it, the agent **1922** requests a new working memory from the rule engine. As before, the working memory will hold all of the data the agent chooses to assert before and during execution of the rule engine. Once the move operation completes, the old version of the agent **1918** in execution environment **1914** indicates to the requesting application **1918** in execution environment **1912** that the move operation has completed. Once the notification has been made, the old agent **1934** is destroyed.

Referring now to FIGS. **20-21**, diagrams illustrating an exemplary process of moving an agent from one execution environment to another with a fetched as-needed set of canonical rules are shown. An application **2018** in execution environment **2012** requests that an agent **2022** in execution environment **2014** move to execution environment **2016**. The location of execution environment **2016** may be described in the move request by an IP address and port, Uniform Resource Locator (URL), or any other means of addressing. The move request includes a reference to a rule repository **2038** from which the agent should fetch a new as-needed rule set. Upon receiving the move request, the agent **2022** requests a new as-needed rule set from the supplied rule repository **2038** based on its goals and target execution environment **2016**. After receiving the new canonical rule set **2034**, the agent **2022** discards its rule engine **2030** along with the associated compiled rule set **2028** and working memory **2032**. In addition the agent **2022** discards its old canonical rule set **2020**. At this point, the agent **2022** encodes itself along with its new as-needed canonical rule set **2034** into a transferable form **2024**. Though a byte array is shown, the encoded agent could take any form that can be transferred between the two execution environments. Once the agent **2022** has created an encoded version of itself **2024** in execution environment **2014** it transfers the encoded version **2024** to an agent manager **2026** residing in execution environment **2016**.

Referring now to FIG. **21**, the process continues with the agent manager **2122** receiving an encoded agent **2134**. Upon receipt of the encoded agent **2134**, the agent manager **2122** decodes the encoded agent **2134** into a new version of the agent **2124** and its new canonical rule set **2126** in execution environment **204**. Once the agent **2124** and rule set **124** have been materialized, the agent manager **2122** requests that the agent **2124** initialize. This request prompts the agent **2124** to go to the execution environment's rule compiler **2120** and request compilation of its canonical rule set **2126**. The result is a compiled rule set **2128**. The agent then creates a new rule engine **130** and subsequently passes the compiled rule set **2128** to it. As during construction, if the execution environment has a sharable rule engine, then one may not need to be created. Once the engine **2130** has been located/created and the compiled rule set **2126** has been added to it, the agent **2124** requests a new working memory from the rule engine. As before, the working memory will hold all of the data the agent chooses to assert before and during execution of the rule engine. Once the move operation completes, the old version of the agent **2138** in execution environment **2114** indicates to the requesting application **2118** in execution environment **2112** that the move operation has completed. Once the notification has been made, the old agent **2138** is destroyed.

Dynamic Determination of Needed Rules

Large rule sets, even with efficient algorithms such as Rete, are often expensive in computation and bandwidth. The process of dynamically removing rules considered unlikely to be useful has a benefit to performance and also, combined with mobile agents, provides an efficient method for utilizing large rule sets that can be partitioned across many repositories. This method also allows an agent to dynamically change the rules to meet the execution environment processing task.

Each constructed agent has a unique identifier for itself and this identifier is also known to the agent's originator. At the point of origination, this identifier will be associated with the agent's outcome. An example outcome is successfully attaining an end goal and sending the results back to the application. Another example outcome is the loss or death of the agent. An agent that is determined to be lost or dead may cause a replacement agent to be launched. The replacement agent will have a unique identifier that differs from the original agent. In addition to a unique agent identifier, an agent also carries with it an indicative subset of the set of previously completed agent outcomes for the given domain. This is a set of unique identifiers and outcomes for agents that have previously executed in the domain of the current agent.

In each execution environment, the local rule repository not only stores rules, but is also the location for agents to record statistics about rule engine activity for the rules in the rule set given to the rule engine. These instrumented rules include agent carried rules and rules for the domain that were retrieved from the local rule repository. Alternately, only the locally acquired domain rules may be instrumented.

Referring now to FIG. **22**, a diagram illustrating an exemplary process of a rule-based agent updating rule statistics when rule processing has completed in an execution environment is shown. As before, an agent **2218** starts its associated rule engine **2222** to process its compiled rule set **2220**. During the course of execution, the rule engine **2222** may successfully match part of the condition (left hand side) of a rule, may match the condition of a rule and activate it, or may match and activate and fire a rule (perform the consequences of the rule). A rule engine may provide for collection of the statistics for the phases of rule activity mentioned. Alternately, the agent may integrate listener code to monitor these phases of rule execution and collect the statistics as the rule engine executes. A rule being fired may result in the rule asserting new data into the working memory **2224** and/or the agent **2218** collecting more data and asserting that into the working memory **2224**. Once an end goal terminates rule processing, or the agent receives a move event, a termination event, a timeout or some other event, then the rule engine is halted. At this point, the agent **2218** requests rule statistics from the rule engine **2222** or collects the statistics from the agent's rule engine listener. These statistics may include, but are not limited to the number of times a rule was fired, the number of times a rule was activated, the number of times a goal in the condition of a rule was matched, the number of times a part of the condition of a rule was matched, or any combination of the above. The statistics **2226** are then added to aggregate rule history stored in the local rule repository **2216**. These stored statistics may include statistics for rules that are not available in the local rule repository since an agent can carry rules with it as it moves.

When the agent prepares to move to another execution environment it dynamically determines to remove unnecessary rules by consulting the rule history associated with some or all of the rules in its current rule set in conjunction with the indicative subset of previously completed agent outcomes that the agent carries. Referring now to FIG. **23**, a diagram illustrating an exemplary process of a rule-based agent dynamically removing unnecessary rules as part of movement to another execution environment is shown. An application **2318** requests that an agent **2326** in execution environment **2314** move to execution environment **2316**. The agent **2326** requests a set of rules from the local rule repository **2322** that are allowed to be carried to other platforms. The result is a canonical rule set **2334**. This rule set is then merged with the set of rules **2320** that the agent **2326** carried with it to execution environment **2314**. The result is canonical rule set **2336**.

At this point the agent consults the local rule repository **2322** to get the rule history **2330** of the rules in set **2336**. The agent **2326** then uses the rule history **2330** with its carried set of previous agent outcomes to remove rules from rule set **2336** that are unlikely to participate in a desired outcome. The statistics are used in aggregate form. As an example consider an agent that carries the results of **2318** previously executed agents and their outcomes, 50 of which were desirable outcomes. The agent examines the metrics for a particular rule named "A" which shows that it was never activated. The agent then removes rule "A" from its agent carried rule set. As another example consider rule "B" which has been activated and fired in one-third of previous desirable outcomes but also has been active and fired in nearly all negative outcomes. Rule "B" remains in the agent carried rule set. Finally, a rule, "C", which never activates for any as yet recorded desired outcomes but has been active in almost all negative outcomes can be considered a computational burden and removed from the agent carried rule set. Although activation is a criterion above, finer grained partial left-hand side matching statistics can be used as well. Since rule removal requires an aggregate of previous runs a threshold is provided so that no rule deletion is permitted until a requisite number of outcomes has been obtained.

Once the pruned rule set **2332** has been created, the agent **2326** encodes itself along with its pruned rule set **2332** into a transferable form in execution environment **2314**. The agent **2326** then transfers the encoded version of itself in execution environment **2314** to an agent manager **2346** resident in the target execution environment **2316**. The remainder of the move process follows that of FIG. **5**.

Survivability Rules

All agents have a lifespan; but that lifespan need not be pre-determined if a set of rules around survivability of an agent is put in place. These rules may be agent specific or execution environment specific. They may be carried with the agent or resident in a rule repository for the execution environment. As these rules are like any other declarative rules, they may be any combination of the above according to the teachings of this invention. In addition, these rules may be used in conjunction with more typical survivability mechanisms such as heartbeats between the application and the agent.

Referring now to FIG. **24**, a diagram illustrating an exemplary process of an agent using a set of survival rules to determine its lifespan is shown. Agent survivability is controlled by the rules loaded in the local compiled rule set **2428**. As before, the local rule set may be comprised of rules supplied or fetched from rule repository **2420** during construction, rules carried from other visited execution environments and/or execution environment specific rules retrieved from rule repository **2426**. Many sources of data that may be asserted into the working memory and, combined with the local rule set **2428**, affect the agent's **2424** lifespan. Examples include lifespan update events from application **2418**, heartbeat events from application **2418**, timer events from the execution environment's timer system **2434**, and even state change events from the agent **2424** itself. As data is asserted into the working memory, the rules engine guarantees that applicable rules are fired. Any number of rules might result in the agent **2424** taking actions that affect its survivability. This includes death of the agent **2424** which is shown. When an agent **104** dies it halts rule engine processing, records any collected historical statistics for the local rule set and stores these in the rule repository **2436**.

Data Narrowing Rules

Agent may visit many execution environments each with differing levels of network connectivity or an execution environment with multiple levels/types of network connectivity. Given this, it is important that an agent take this into consideration when responding to application requests, sending periodic reports, and determining how much data to carry with it when moving. As per the teachings of this invention, execution environment specific rules are an ideal method for insuring the appropriate agent behavior. If the networking capabilities of the execution environment are static, then rules for this may be maintained in the rule repository on the execution environment running the application that launched the agent. In many cases though, the capabilities may be more dynamic in which case the rules regarding network bandwidth are better kept on the remote execution environment.

Referring now to FIG. **25**, a diagram illustrating an exemplary process of the of an agent using a set of data narrowing rules to determine how much data should be sent over the network is shown. This diagram shows the same agent in three different scenarios. As before, each agent is communicating with an application **2532** that in this case is hosted on server **2530** which is connected to a high-speed data network, **2534**. In the first scenario, the agent **2514** has been constructed on or moved to server execution environment **2512**, which is connected to the high speed data network directly via a gigabit ethernet link **2544**. The agent **2514** utilized a rule-based system that is driven by the associated rule engine **2516**. This engine **2516** has been loaded with execution environment specific rules about the current network bandwidth capabilities of the execution environment **2512**. In this example the agent **106** completes a task which will ultimately generate a report back to the application **2532** on execution environment

2530. When that task completes, that event causes a rule to fire in the engine 2516, which instructs the agent 2514 to send a detailed report. In this case, a detailed report is appropriate because a high bandwidth connection is available between the agent 2514 and the application 2532.

In the second scenario, that same agent now labeled 114 has moved to a home computer 2518 which is connected to the network via a DSL connection 2546. As before, the engine 2522 is loaded with the execution environment specific rules regarding bandwidth available to the execution environment. As the agent 2520 completes its task, the event causes a rule to fire, which instructs agent 2520 to send a full report, which contains less data than the detailed report described previously. Note, that the agent 2520 is not compressing the same data, but sending a different data-set back—a subset of the data to fit the bandwidth available.

In the final scenario, the agent, now labeled 2526 has moved to the mobile device 2524. The mobile device is connected to the high speed data network via a relatively low speed cellular data network 2536. As before, the agent 2526 completes its task which results in the rule engine 2528 firing a rule. This firing causes the agent 2526 to dispatch a much smaller summary report to the application 2532 in order to accommodate the low bandwidth connection.

Methods, computer readable media and systems have been shown and/or described in the above embodiments for movement of an agent that utilizes a compiled set of canonical rules. Although the above descriptions set forth embodiments, it will be understood that there is no intent to limit the invention by such disclosure, but rather, it is intended to cover all modifications and alternate implementations falling within the spirit and scope of the invention. For example, the present invention should not be limited to a single agent, or to a particular programming language for the execution environment. Furthermore, the association of agent to execution environments is not limited to the topology depicted. Lastly, the embodiments are intended to cover capabilities and concepts whether they be via a loosely couple set of components or they be converged into one or more integrated components, devices, circuits, and/or software programs.

What is claimed is:

1. An apparatus comprising:
- a memory; and
- a processor operatively coupled to the memory, wherein the processor is configured to:
  - collect a set of canonical survival rules associated with a first execution environment and associated with a mobile agent;
  - compile the set of canonical survival rules;
  - execute a rule engine of the mobile agent with the set of canonical survival rules;
  - store a heartbeat from an originating application in a working memory associated with an agent;
  - fire a survival rule from the set of canonical survival rules based upon the heartbeat being stored in the working memory;
  - assert survival data into the memory as a result of firing the survival rule;
  - receive a request to move the mobile agent to a second execution environment;
  - determine a first rule from the first set of canonical survival rules has never fired;
  - determine a second rule from the first set of canonical survival rules has been fired;
  - determine a third rule from the first set of canonical survival rules has been active in only negative outcomes, wherein the first rule and third rule are removed to create a filtered set of canonical survival rules;
  - encode the mobile agent with the filtered set of canonical survival rules; and
  - transmit the encoded mobile agent to the second execution environment, wherein the second execution environment decodes and executes the mobile agent, wherein the mobile agent uses the filtered set of canonical survival rules in the second execution environment.

2. The apparatus of claim 1, wherein the set of canonical survival rules is specific to the mobile agent.

3. The apparatus of claim 1, wherein the set of canonical survival rules is specific to an execution environment of the apparatus.

4. The apparatus of claim 1, wherein the survival data is asserted into the memory based at least in part on a state change of the mobile agent.

5. The apparatus of claim 1, wherein the survival data is asserted into the working memory based at least in part on a timer event in an execution environment of the apparatus.

6. The apparatus of claim 1, wherein the processor is further configured to request a second set of canonical survival rules based upon the received request to move the mobile agent to a second execution environment.

7. The apparatus of claim 6, wherein the processor is further configured to merge the second set of canonical survival rules with the first set of canonical survival rules.

8. A method comprising:
- collecting, by a processor of a computing device, a set of canonical survival rules associated with a first execution environment and associated with a mobile agent;
- compiling the set of canonical survival rules;
- executing a rule engine of the mobile agent with the set of canonical survival rules;
- storing a heartbeat from an originating application in a working memory associated with an agent;
- firing a survival rule from the set of canonical survival rules based upon the heartbeat being stored in the working memory;
- asserting survival data into a memory of the computing device as a result of firing the survival rule;
- receiving a request to move the mobile agent to a second execution environment;
- determining a first rule from the first set of canonical survival rules has never fired;
- determining a second rule from the first set of canonical survival rules has been fired;
- determining a third rule from the first set of canonical survival rules has been active in only negative outcomes, wherein the first rule and third rule are removed to create the filtered set of canonical survival rules;
- encoding the mobile agent with the filtered set of canonical survival rules; and
- transmitting the encoded mobile agent to the second execution environment, wherein the second execution environment decodes and executes the mobile agent, wherein the mobile agent uses the filtered set of canonical survival rules in the second execution environment.

9. The method of claim 8, wherein the set of canonical survival rules is specific to the mobile agent.

10. The method of claim 8, wherein the set of canonical survival rules is specific to an execution environment of the computing device.

11. The method of claim **8**, wherein the survival data is asserted into the working memory based at least in part on a state change of the mobile agent.

12. A tangible non-transitory computer-readable medium having computer-readable instructions stored thereon, the computer-readable instructions comprising:

instructions to collect a set of canonical survival rules associated with a first execution environment and associated with a mobile agent;

instructions to compile the set of canonical survival rules;

instructions to execute a rule engine of the mobile agent with the set of canonical survival rules;

instructions to store a heartbeat from an originating application in a working memory associated with an agent;

instructions to fire a survival rule from the set of canonical survival rules based upon the heartbeat being stored in the working memory;

instructions to assert survival data into a memory of a computing device as a result of firing the survival rule;

instructions to receive a request to move the mobile agent to a second execution environment;

instructions to determine a first rule from the first set of canonical survival rules has never fired;

instructions to determine a second rule from the first set of canonical survival rules has been fired;

instructions to determine a third rule from the first set of canonical survival rules has been active in only negative outcomes, wherein the first rule and third rule are removed to create the filtered set of canonical survival rules;

instructions to encode the mobile agent with the filtered set of canonical survival rules; and

instructions to transmit the encoded mobile agent to the second execution environment, wherein the second execution environment decodes and executes the mobile agent, wherein the mobile agent uses the filtered set of canonical survival rules in the second execution environment.

13. The non-transitory computer-readable medium of claim **12**, wherein the survival data is asserted into the working memory based at least in part on a state change of the mobile agent.

14. The non-transitory computer-readable medium of claim **12**, wherein the survival data is asserted into the working memory based at least in part on a timer event in an execution environment of the computing device.

15. The non-transitory computer-readable medium of claim **12**, wherein the set of canonical survival rules is specific to an execution environment of the computing device.

16. The non-transitory computer-readable medium of claim **12**, wherein the instructions further comprise instructions to request a second set of canonical survival rules based upon the received request to move the mobile agent to a second execution environment.

17. The non-transitory computer-readable medium of claim **16**, wherein the instructions further comprise instructions to merge the second set of canonical survival rules with the first set of canonical survival rules.

* * * * *